METEORIC PROGRAMMING

For the ORIC-1

JOHN VANDER REYDEN

METEORIC PROGRAMMING

For the ORIC-1

METEORIC PROGRAMMING

For the ORIC-1

JOHN VANDER REYDEN

MELBOURNE HOUSE

Published in the United Kingdom by:
Melbourne House (Publishers) Ltd.,
Glebe Cottage, Glebe House,
Station Road, Cheddington,
Leighton Buzzard, Bedfordshire. LU7, 7NA.

ISBN 0 86161 124 1

Published in Australia by:
Melbourne House (Australia) Pty. Ltd.,
Suite 4, 75 Palmerston Crescent,
South Melbourne, Victoria, 3205.

National Library of Australia Card Number and
ISBN 0 86759 121 8

Published in the United States of America by:
Melbourne House Software Inc.,
233 South Beverly Drive,
Beverley Hills,
California. 90212

All programs in this book are
Copyright (c) 1983 Beam Software

This book is of course not a solo effort and I would like to acknowledge the generous help provided by: Craig McFarlane, Bruce Bayley, Kevin Tobin, Clive Barrett, Denver Jeans and Peter Falconer in making this book a reality.

All rights reserved. This book is copyright. No part of this book may be copied or stored by any means whatsoever whether mechanical or electronic, except for private or study use as defined in the Copyright Act. All enquiries should be addressed to the publishers.

Printed in Singapore by Richard Clay (S.E.Asia) Pte Ltd

Publisher's Note

In keeping with our ongoing commitment to provide both literature and software for personal computers, Melbourne House is very proud to be able to publish this book of games and general routines for the Oric-1.

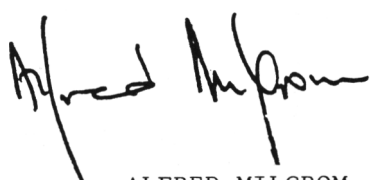
You will find complete program listings, comprehensive structures and useful hints on each program. We have taken care to design the format in such a way that the programs will be easily read, to reduce the possibility of transcription errors, especially with the graphics characters.

After working through this book, I think you will agree that the programs should set the standard by which future programs for the Oric-1 will be judged by.

I know you will enjoy not only the programs themselves, but also the insight you will gain about programming the Oric-1.

Melbourne House are not just publishers - we are dedicated to microcomputer software, and we are always interested in your feedback: If you have an article or program that you think might be of value to other users we want to hear from you.

Until I hear from you, let me just wish you happy programming!



ALFRED MILGROM
PUBLISHER

SPECIAL NOTE

You may have noticed that the listings in the book don't look exactly like the listings you normally see on your screen.

When a program is listed or printed out on the ORIC the keywords are all compressed. The problem with this is that it makes it very hard for anyone to read the program, and harder still if you have to read a part of a line, key it in, and still know exactly where you were in the program line.

To overcome this problem and make it easier for you to key the listings in we have inserted spaces on either side of all colons(:), semi-colons(;), and commas(,) as well as any other places where there could be difficulty in reading the listing. For example, a normal listing would look like this:

```
10 CURMOV6,0,3:PRINTA$;"-";
```

The same line in the book would look like this:

```
10 CURMOV6 , 0 , 3 : PRINT A$ ; "-" ;
```

When you key the programs in you should not key any spaces in other than where they occur within quotation marks.

If you find that you can't enter a whole line go back and check that you have not entered any spaces that should not be there.

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#)

Contents

GENERAL PROGRAMS

Leapfrog 1

Asteroids in Space 4

Oriclock 6

Draw 10

3D Mazeman 15

EDUCATIONAL

Bubble Sort 22

Sort 1

Sort 2

Sort 3

Sound Aid/Synthesizer 29

MATHEMATICAL

Simultaneous Equations 35

3D Plot 40

BUSINESS

Payroll 42

Sales Analysis 50

ARCADE GAMES

Space Escape 55

Bomber 59

Freeway Frog 65

Duck Shoot 74

Alien Blitz 78

Chopper 83

Luna Landa 89

Meteor Storm 94

Eliminator 102

ORIC Invaders 110

North Sea Copter 117

Adventure 123

UTILITY PROGRAMS 137

ORIC-1's BASIC 138

Machine Code Monitor 144

Renumber 149

Token Table 163

Block Line Deletion 166

Graphics Aid 173

Notes for Programmers

The following notes are meant to be read by all persons using the programs in this book. Some are just tips from our own experience and some are things that you won't find in the manual.

KEYBOARD SCANNING

As you may know the KEY$ function on the ORIC does not allow for the keys to repeat (i.e. to get KEY$ to recognize a key more than once you must release the key and press it again). This is not very useful for games where you are trying to move and fire your ship (or whatever). You can overcome this problem by peeking a location in ORIC's memory which changes according to the key being held down. The memory location is 520 decimal.

The following table may save you some trouble.

CHARACTER	PEEK (520)	ASCII
	172	8
	188	9
	180	10
	156	11
space	132	32
null	56	error

See also Sound Aid (Play mode) for an advanced method of keyboard scanning.

CONTROL CHARACTERS

Control characters are used to determine the mode of certain operations such as Caps lock, keyclick, double height, etc. The only problem with control characters is that they toggle. This means that if a program uses a control character, and the program is re-run, then the control will toggle to its unwanted state. To overcome this, you can poke a value directly into the memory location which controls these modes. This location is 618 decimal. The most common setting is Cursor-off and keyclick-off. This is achieved by the statement POKE 618,10. Other values may be found by setting the states using control characters then typing PRINT PEEK (618).

USING CURMOV

You cannot use CURMOV without first having set CURSET.

ERROR MESSAGES

If you crash a program and can't see/read the error message just type Paper (n). The screen goes back into text mode and displays the error.

CURSOR POSITION

You may find, when programming, that printing numbers on the screen with the PLOT command can be messy or you may find it difficult to position an INPUT statement. These and other problems can be overcome with the ability to position the PRINT cursor anywhere on the screen and using the PRINT statement.

The programming method is as follows:

```
POKE 616, vertical position
PRINT
POKE 617, horizontal position
PRINT anything
```

Using the same principle a downward screen scroll may be achieved as follows:

```
FOR K = 1 to 25
POKE 616, 255
PRINT
NEXT K
```

CSAVE

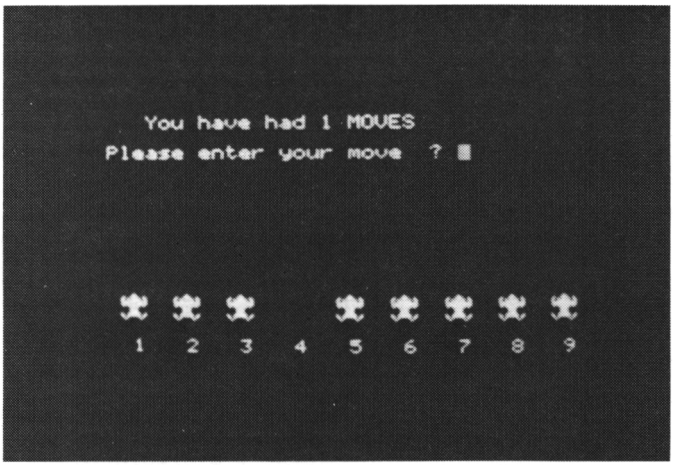
We recommend that you save all the programs before running them. To be on the safe side, and to make sure you don't have to start keying in again, we advise you to use the slow rate. Once you have a "slow" master copy then make a fast one for normal use.

STR$ FUNCTION

When using this function don't forget that the first character in the string will be used for the sense (+, -) of the numerical value. This can also affect the colour used if you print the string.

e.g. STR$ (8)= "STX 8"= 2 characters

Leapfrog



This game makes a good mind exercise. The idea is simple. Move all the frogs from the left side to the right side and vice-versa by hopping over only one frog at a time. If you succeed, see if you can do it again in less moves.

STRUCTURE

FUNCTION	LINE(s)
Initialization	100 - 170
Display set-up	180 - 220
Prompt for move	240
Test if move valid	250 - 280
Move frogs	300 - 330
Game-end routine	350 - 410
Position-INPUT routine	420 - 450
Create frog characters	430 - 520
Frog character data	460 - 490

VARIABLES

A1$ = Left set of frogs
A2$ = Right set of frogs
CN = Move-counter
B1$ = Frog being moved
B2$ = Frog being replaced
k$ = Input from player

```
100 PAPER0 : INK2
110 GOSUB 430
120 A1$=CHR$(2)+"AB " : A1$=A1$+A1$+A1$+A1$+"    "
130 A1$=A1$+CHR$(3)+"AB "+CHR$(3)+"AB "+CHR$(3)+"AB
    "+CHR$(3)+"AB"
140 A2$=CHR$(2)+"CD " : A2$=A2$+A2$+A2$+A2$+"    "
150 A2$=A2$+CHR$(3)+"CD "+CHR$(3)+"CD "+CHR$(3)+"CD
    "+CHR$(3)+"CD"
160 B1$=A1$ : B2$=A2$
170 CN=0
180 CLS : PLOT3 , 23 , "1   2   3   4   5   6   7   8
    9"
190 PLOT1 , 20 , B1$ : PLOT1 , 21 , B2$
200 IFLEFT$(B1$ , 15)=RIGHT$(A1$ , 15)ANDRIGHT$(B1$ ,
    15)=LEFT$(A1$ , 15)THENGOTO350
210 PLOT1 , 8 , "   You have had"+STR$(CN)+" moves"
220 PLOT1 , 10 , "Please enter your move      "
230 GOSUB420
240 INPUT K$
250 IF LEN(K$)<>2THENGOTO210
260 FR=4*(ASC(LEFT$(K$ , 1))-49)+1
270 TR=4*(ASC(RIGHT$(K$ , 1))-49)+1
280 IF(MID$(B1$ , TR ,
    1)<>" "ORABS(TR-FR)>8)THENGOTO210
290 CN=CN+1
300 B1$=LEFT$(B1$ , TR-1)+MID$(B1$ , FR ,
    3)+MID$(B1$ , TR+3)
310 B2$=LEFT$(B2$ , TR-1)+MID$(B2$ , FR , 3)+MID$(B2$
    , TR+3)
320 B1$=LEFT$(B1$ , FR-1)+"  "+MID$(B1$ , FR+3)
330 B2$=LEFT$(B2$ , FR-1)+"   "+MID$(B2$ , FR+3)
340 GOTO190
350 PLOT3 , 8 , "You did it in"+STR$(CN)+" MOVES"
360 PLOT1 , 10 , " Do you want another go?       "
370 GOSUB420
380 GETK$
390 IF K$="Y"THENGOTO160
400 IF K$<>"N"THENGOTO360
410 CLS : END
420 POKE618 , 10 : POKE616 , 10 : PRINT : POKE617 ,
    26 : POKE618 , 3 : RETURN
430 FORI=46600TO46631
440 READX : POKEI , X
450 NEXTI
460 DATA1 , 19 , 53 , 39 , 63 , 63 , 23 , 3
470 DATA32 , 50 , 43 , 57 , 63 , 63 , 58 , 48
480 DATA3 , 3 , 39 , 55 , 20 , 8 , 0 , 0
490 DATA48 , 48 , 57 , 59 , 10 , 4 , 0 , 0
500 PLOT1 , 20 , CHR$(9)
510 PLOT0 , 21 , CHR$(9)
520 RETURN
```

4#

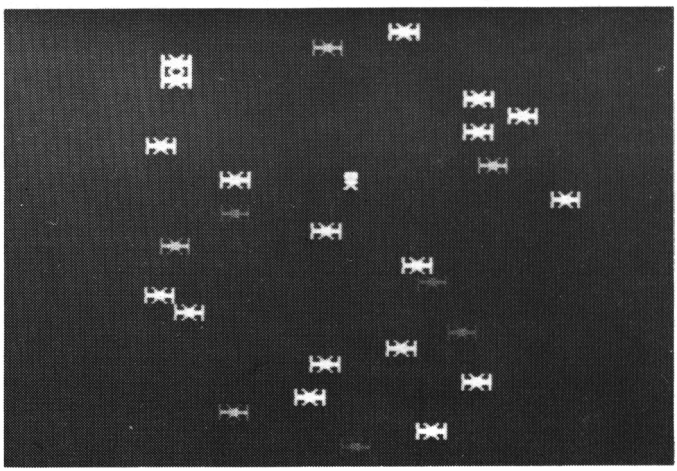
1#

3#

3#

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

Asteroids in Space



You are travelling through space when you encounter a space storm. You must avoid the asteroids by moving left and right using the "←" and "→" keys. After some time you end up in the heart of the storm and you must negotiate your way between much larger asteroids. Eventually you will crash, when you do you will be given a survival rating.

HOW THE PROGRAM WORKS

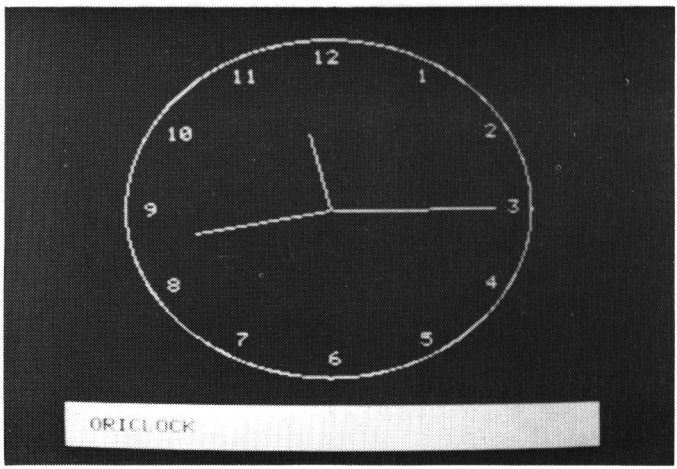
This program uses the PRINT function to scroll the screen upwards. When the display is scrolled, your ship will move with everything else, so it is overwritten with spaces then printed again in the correct position. In this way your ship stays on the same line while everything else moves. The creatures and space ships appear randomly at the bottom of the screen. The screen is scrolled two lines each cycle so that there are not too many obstacles on the screen at any one time.

IMPROVING THE PROGRAM!

The program has been kept deliberately simple to enable you to improve it. Firstly, you can create more stages using the user defined character set on the Oric. The test for the second stage is in line 270. Secondly, more sound could be included, if only to slow the program up somewhat. You can slow the program by changing the length of the notes in your sounds. Thirdly, you could include the option to fire at the objects on the screen, adding their values to your score, and so on!

```
100 REM * ASTEROIDS *
110 REM
120 PAPER0 : INK7 : CLS
130 PRINT CHR$(17)
140 PLOT 15 , 2 , "FIND THE GAP...."
150 PLOT 15 , 3 , "ASTEROIDS IN SPACE"
160 FOR J=35 TO 38
170 FOR I=0 TO 7
180 READ K
190 POKE 46080+I+(J*8) , K
200 PLOT P , X , "&"
210 NEXT I
220 NEXT J
230 A$="#"
240 N=0
250 T=1
260 P=19 : X=10
270 R=INT(RND(1)*35)+3
280 PLOT R-1 , 26 , CHR$(INT(RND(1)*7)+1)
290 PLOT R , 26 , A$
300 PLOT P , X-2 , " "
310 PLOT P , X , "&" : PLOT P-1 , X , CHR$(2) :
    PLOT P+1 , X , CHR$(0)
320 N=N+T
330 IF N=100 THEN A$="$%"
340 IF N=104 THEN T=2
350 PLOT P , X-1 , " "
360 PLOT P , X , "&"
370 IF SCRN(P , X+2)<>32 THEN 450
380 IF SCRN(P+T , X+2)<>32 THEN 450
390 IF SCRN(P-T , X+2)<>32 THEN 450
400 X$=KEY$
410 IF X$="W" THEN PLOT P , X , " " :
    P=P-T : IF P<3 THEN P=37
420 IF X$="E" THEN PLOT P , X , " " : P=P+T :
    IF P>37 THEN P=3
430 PRINT : PRINT
440 GOTO 270
450 EXPLODE
460 PRINT"your survival rating ";N
470 PRINT CHR$(17)
480 PRINT"do you want another game , Y/N" :
    INPUT A$ : IF A$="Y"THEN RUN
490 DATA 45 , 45 , 30 , 61 , 51 , 30 , 33 , 63
500 DATA 36 , 34 , 35 , 63 , 63 , 35 , 34 , 36
510 DATA 9 , 17 , 49 , 63 , 63 , 49 , 17 , 9
520 DATA 31 , 21 , 31 , 4 , 31 , 4 , 10 , 17
530 END
```

Oriclock



This program is a simulation of a REAL TIME CLOCK. The clock has alarm facilities and ticks away for as long as you like to leave it. The alarm is bound to wake you (by driving you mad). If you find it is still not loud enough then increase the volume if you wish.

The clock is not extremely accurate as it uses the computer's "wait" instructions. You can adjust this wait statement to tune the clock's accuracy though it will always tend to vary in cycle time. The clock will stop on alarm, this is unavoidable.

The Oriclock is not meant to replace your clock but is a demonstration program with visual appeal and purpose. If anyone knows how to (if it is possible) get access to the frames controller (sends picture "frames" to video output) we would like to know as this will provide a stable oscillator to use to control the clock's accuracy.

HOW TO RUN THE PROGRAM

Type RUN followed by (ENTER) to start the program after having loaded it. You will need to enter the HOUR (12 or less), MINUTE and SECONDS at which you want the clock to start. The program checks the validity of your input to prevent you from entering wrong data. You will also be asked if you wish to set the alarm, to do so you enter the values you require in the same format as above.

ORICLOCK STRUCTURE:

INITIALIZATION:
1. Display clock-face.
2. Input hour, minute and seconds.
3. If alarm set;
4. Set alarm hours, minutes and seconds.
5. Initialize starting time.

THE MAIN LOOP:
H. Determine hour hand position
M. Determine minute hand position
S. Determine second hand position
   Draw second, minute hour hands
   Display clock face
   If alarm time matches real time then alarm rings
   Wait 1 second
   Re-initialize time
   Draw over second hand
   If same minute then goto S:
   Draw over minute hand
   If same hour then goto M
   Draw over hour hand
   Goto H

HOW THE PROGRAM WORKS:

Sin and Cos functions are used to draw the clock hands. The second hand is moved every second, the minute hand every minute and the hour hand every 12 minutes.

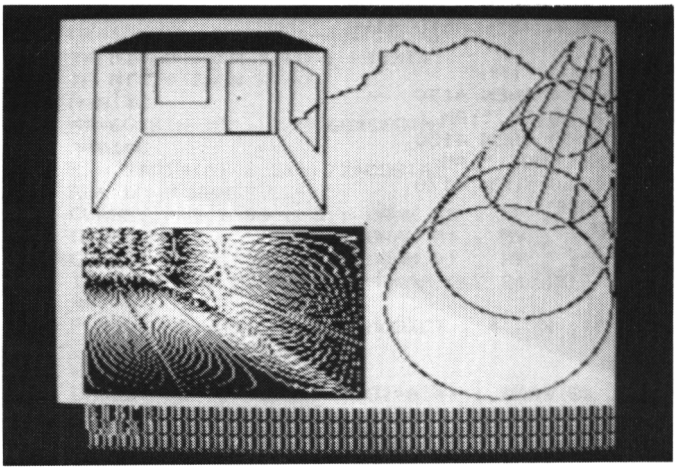
CONSIDERATIONS:

Due to the fact that the clock is only using the "wait" instruction for timing, the clock is not very accurate. Therefore to get any reasonable accuracy. It would be necessary to divide into the clock timer chip within ORIC.
Line 1160 would then become:
1160 IF INT (TIMER/FREQ) < = 0 THEN 1160
where freq = 50 or 60 i.e. frequency of mains

```
   5 REM * ORICLOCK *
  10 CLS : PAPER1 : INK0 : N=1 : T=30
  20 GOTO 3000
1000 REM CLOCK MOVE
1005 PAPER1 : INK0
1020 IF HI=60 THEN HI=0
1040 H=HI*C
1050 HX=40*SIN(H) : HY=-40*COS(H)
1060 IF MI=60 THEN MI=0 : HT=HT+1
1065 IF HT=24 THEN HT=0
1070 M=MI*C
1080 MX=60*SIN(M) : MY=-60*COS(M)
1090 A=SI*C
1100 SX=72*SIN(A) : SY=-72*COS(A)
1110 REM DRAW HANDS
1120 CURSET 119 , 96 , 3 : DRAW SX , SY , 1
1130 CURSET 119 , 96 , 3 : DRAW MX , MY , 1
1140 CURSET 119 , 96 , 3 : DRAW HX , HY , 1
1155 IF(AL=1 AND HT=AH AND MI=AM AND SI=AS)THEN 2000
1157 N=7-N
1158 PLAY1 , 0 , 1 , 100 : MUSIC1 , 4 , N , 5 : PLAY0
     , 0 , 0 , 0
1160 WAIT(T)
1210 CURSET 119 , 96 , 3 : SI=SI+1 : DRAW SX , SY , 2
1220 IF SI=60 THEN SI=0 ELSE 1090
1230 CURSET 119 , 96 , 3 : MI=MI+1 : DRAW MX , MY , 2
1240 IF (MI-INT(MI/12)*12)=0 THEN HI=HI+1 ELSE 1060
1250 CURSET 119 , 96 , 3 : DRAW HX , HY , 2
1260 GOTO 1020
2000 PING
2010 A0$=KEY$
2020 IF A0$=" " THEN 1210
2025 WAIT 100
2030 GOTO 2000
3000 REM SET CLOCK
3030 P1=3.14159 : C=P1/30
3040 HIRES : CURSET 119 , 96 , 3 : CIRCLE 90 , 1
3050 FOR I=1 TO 12
3060 Q=I/6*PI
3070 CURSET 118+80*SIN(Q) , 92-80*COS(Q) , 3
3080 IF I<10 THEN CHAR 48+I , 0 , 1 : GOTO 3120
3090 IF I=10 THEN CHAR 49 , 0 , 1 : CURMOV 6 , 0 , 3 :
     CHAR 48 , 0 , 1 : GOTO 3120
3100 IF I=11 THEN CHAR 49 , 0 , 1 : CURMOV 6 , 0 , 3 :
     CHAR 49 , 0 , 1 : GOTO 3120
3110 CURMOV-5 , 0 , 3 : CHAR 49 , 0 , 1 : CURMOV 6 ,
     0 , 3 : CHAR 50 , 0 , 1
3120 NEXT I
4000 INPUT"HOUR.";HT
4010 IF HT<0 OR HT>24 THEN 4000
4020 INPUT"MINUTES";MI
4030 IF  MI<0 OR MI>60 THEN 4020
4040 INPUT"SECONDS";SI
```

```
4050 IF SI<0 OR SI>60 THEN 4040
4060 IF HT>12 THEN HT=HT-12 ELSE HI=HT
4070 HI=HI*5+INT(MI/12)
4080 CLS : PRINT"DO YOU WANT TO SET THE ALARM." :
     PRINT"(Y OR N)"
4090 INPUT A$
4100 IF A$<>"Y" AND A$<>"N" THEN 4090
4110 IF A$="N" THEN AL=0 : GOTO 4190
4120 AL=1 : CLS
4130 INPUT"ALARM HOUR.";AH
4140 IF AH<0 OR AH>24 THEN 4130
4150 INPUT"ALARM MINUTES.";AM
4160 IF AM<0 OR AM>60 THEN 4150
4170 INPUT"ALARM SECONDS.";AS
4180 IF AS<0 OR AS>60 THEN 4170
4190 REM DISPLAY TIME
4200 PRINT : PRINT
4210 PRNT"ORICLOCK."
4220 GOTO 1000
4230 END
```

Draw



This is a program that allows you to "draw" on the screen from the keyboard, using the cursor control keys to determine the direction of the cursor. It draws a pixel at a time and you can change direction, one pixel at a time. You also have a CIRCLE command that will draw a circle with a specified radius from the cursor position.

The program starts in TEXT mode and displays the options available. The options are as follows :

1) BACKGROUND
Allows choice of background colour.

2) DRAW-ON/OFF
Useful when you want to move the cursor without drawing on the display.

3) COLOUR CHANGE
Allows user to change the colour of the foreground while drawing. Changing the foreground colour changes everything drawn (i.e. only one foreground colour on the screen at any time).

4) CIRCLE
The position of the cursor is tested and the maximum radius is calculated and displayed to the user. Only circles up to this radius may be drawn while the cursor is in this position. The circle is drawn in INVERSE so if a circle is drawn twice in the same place, it will remove itself.

5) PAINT

Used for filling in objects. The routine is written in Basic so is very slow though you may exit during a PAINT by hitting any key. The method used for painting is as follows : The cursor moves to the left until it meets a foreground colour and then turns right. It then 'hugs' to the left and keeps travelling around the perimeter of the object until it is completely surrounded by foreground.

STRUCTURE

FUNCTION	LINE(s)
Initialization	10 - 195
Keyboard input	190 - 320
Paint subroutine	3000 - 3130
Vertical movement	390 - 490
Horizontal movement	500 - 600
Colour change	610 - 640
Circle command	700 - 750
PRINT AT routine	2000 - 2020
Cursor sound	1010

VARIABLES

C = Current Colour
NU = ASCII value of control key symbol
X,Y = Position of cursor

SPECIAL NOTES

Logic Flow of PAINT Option :

1) Set cursor to horizontal LEFT

2) Move cursor

3) Pixel under cursor? -NO-(2)

4) Change cursor's direction to vertically UP

5) Pixel on LEFT of cursor? -YES-TURN LEFT-(9)

6) Put a pixel

7) Pixel in front of cursor? -YES-TURN RIGHT-(9)

8) Move cursor -(5)

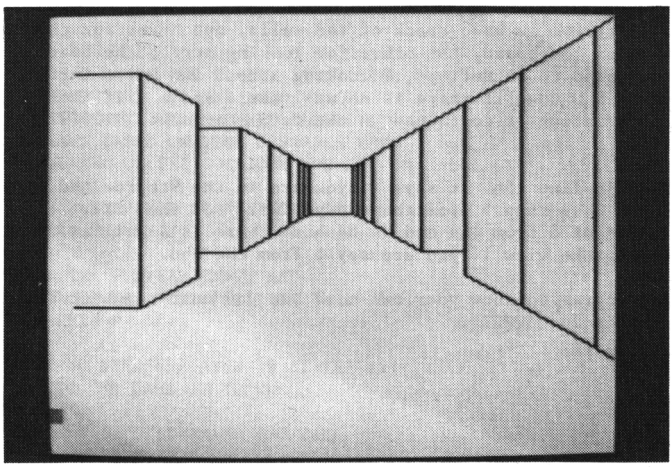
9) Cursor completely surrounded? -YES-RETURN

10) Move cursor -(5)

```
 5 REM * DRAW *
10 CLS  : TEXT
15 PAPER 0  : INK 7  : KB=520
20 FOR I=1 TO 16
30 READ PZ , S$
40 GOSUB 2000  : PRINT S$
50 NEXT
60 DATA 7 , "COLOURS AVAILABLE ARE : " , 39 ,
   0 - BLACK
61 DATA 71 , 1 - RED , 103 , 2 - GREEN
70 DATA 135 , 3 - YELLOW , 167 , 4 - BLUE
72 DATA 199 , 5 - MAGENTA , 231 , 6 - CYAN , 263 ,
   7 - WHITE
78 DATA 327 , "INSTRUCTION SET : "
80 DATA 359 , ARROWS FOR MOVEMENT , 391 ,
   C3-CHANGE TO COLOUR 3 , 423 , N-NO TRACE
90 DATA 455 , "Y-YES ,  TRACE A LINE"
100 DATA 487 , R-CIRCLE RADIUS (1-9)
120 DATA 615 , WHAT COLOUR BACKGROUND ?
140 I$=KEY$  : IF I$=""THEN 140
150 I=VAL(I$)  : IF I=0 AND I$<>"0" THEN 140
160 IF I>7THEN 140
165 INK I
170 B=I : C=B : X=128 : Y=96 : ST=1 : NU=32 : DV=1
180 F=B+1 : IF F>7 THEN F=0
190 CLS : HIRES : PAPER B : INK F
195 POKE 618 , 10 : REM CURSOR & CLICK OFF
200 I$=KEY$ : T=POINT(X , Y) :
   IF T=-1 THEN T=1 ELSE T=0
205 FOR K=0 TO 2 : CURSET X , Y , K : NEXT :
   CURSET X , Y , T
210 IF I$=""THEN 200
220 DE=DV
230 NU=ASC(I$)
240 IF NU=11 THEN 400
250 IF NU=10 THEN 410
260 IF NU=8  THEN 510
270 IF NU=9  THEN 520
280 IF NU=80 THEN 330
290 IF NU=67 THEN 610
300 IF NU=78 THEN ST=-1 : GOTO 200
310 IF NU=89 THEN ST= 1 : GOTO 200
315 IF NU=82 THEN 700
320 GOTO 200
330 REM *** PAINT ***
350 GOSUB 3000
360 PING
380 GOTO 200
390 DE=DV : GOTO 410
400 DE=-DV
410 Y=Y+DE
415 GOSUB 1000
420 IF Y<1 THEN Y=1 : GOTO 200
430 IF Y>199 THEN Y=199 : GOTO 200
450 CURSET X , Y , ABS(2-ST)
470 IF PEEK(KB)=156 THEN 400
```

```
 480 IF PEEK(KB)=180 THEN 390
 490 GOTO 200
 500 DE=DV : GOTO 520
 510 DE=-DV
 520 X=X+DE
 525 GOSUB 1000
 530 IF X<12 THEN X=12 : GOTO 200
 540 IF X>239 THEN X=239 : GOTO 200
 560 CURSET X , Y , ABS(2-ST)
 580 IF PEEK(KB)=172 THEN 510
 590 IF PEEK(KB)=188 THEN 500
 600 GOTO 200
 610 I$=KEY$ : IF I$="" THEN 610
 620 C=VAL(I$) : IF C=0 AND I$<>"0" THEN 610
 630 IF C>7 THEN 610
 635 INK C
 640 GOTO 200
 650 END
 700 REM * CIRCLE *
 705 L=-(X>Y)*Y-(Y>X)*X-(X=Y)*X
 707 IF 239-X<LTHENL=239-X
 710 IF 199-Y<LTHENL=199-Y
 715 PRINT"CIRCLE LIMIT IS  ";L/10
 720 I$=KEY$ : IF I$=""THEN 720
 730 IF I$<"1" OR I$>"9"THEN 720
 740 R=10*VAL(I$) : IF R>LTHEN 700
 750 CIRCLE R , 2 : PRINT : PRINT : PRINT : GOTO200
1000 REM * CURSOR SOUND *
1010 PING : PLAY0 , 0 , 0 , 0 : RETURN
1998 STOP
2000 REM *** PRINT @ ROUTINE ***
2020 YZ=INT(PZ/32) : XZ=INT(PZ-YZ*32) : POKE616 ,
     YZ : PRINT : POKE617 , XZ : RETURN
2998 STOP
2999 REM
3000 REM *** PAINT (Routine) ***
3020 PX=X : PY=Y : XD=-1 : YD=0 : K=0 : KE=50
3030 GOSUB 1000 : PX=PX+XD : IF PX<12 THEN RETURN
3040 IF POINT(PX+XD , PY+YD)=0THEN CURSET PX , PY ,
     3 : GOTO 3030
3045 XD=0 : YD=-1
3050 IF XD=0 THEN XL=YD : YL=0 ELSE XL=0 : YL=-XD
3060 XR=-XL : YR=-YL
3070 IF POINT(PX+XL , PY+YL)=0 THEN XD=XL : YD=YL :
     GOTO 3120
3080 IF PX<13 OR PX>238 OR PY<1 OR PY>198 THEN RETURN
3090 CURSET PX , PY , 1 : K=K+1
3095 IF K>KE THEN 3120
3100 IF POINT(PX+XD , PY+YD) THEN XD=XR : YD=YR :
     GOTO 3120
3110 PX=PX+XD : PY=PY+YD : GOTO 3070
3120 Z=0 : FOR ZX=-1TO1 : FORZY=-1TO1 :
     Z=Z+POINT(PX+ZX , PY+ZY) : NEXT ZY , ZX
3125 IF Z=-9 THEN RETURN
3127 IF KEY$<>""THEN RETURN
3130 K=0 : PX=PX+XD : PY=PY+YD : GOTO 3050
```

3D Mazeman



You are trapped in a maze. You must get out! Claustrophobia seizes you.

You move forward, a passageway appears on the left, quick! Oh no, . . . another dead end.

Will you ever get out of here alive?

This amazing 3D dimensional simulation places you right in the maze, with the task of getting out. The only thing to be thankful for is that there are no monsters here!! Yet!!

You may take as long as you like, but the time you take is being measured. At each position you will see a realistic perspective view. You can turn left or right, move forwards or turn around using the cursor control keys.

If you become hopelessly lost, it might be a good idea to draw a map.

The program will draw what you can see from where you stand in the maze.

When you are close to the exit you will see the 'EXIT' sign. If you choose the correct direction you will find your way out. You will be informed how many steps you took to solve the maze!

HOW THE PROGRAM WORKS

The maze used here is a 9 x 9 grid, somewhat like a chess board. Each position in the maze is represented by a square on the board.

Obviously each square can have walls in any one of the four directions.

This program uses 2 arrays to keep track of the walls, one array for the walls found looking up the board, the other for looking across the board. A total of 18 lines need to be defined, 9 looking across and 9 looking up. For each line we put a 0 down if there is no wall there and a 1 if there is a wall. This information is contained in the data statements 9100-9108, 9200-9208.

The exit is defined in line 156. It says if you are in the 9th row and facing backwards and less than 5 from the end you will see the 'EXIT' sign. The requirement of 5 from the end is because there is a wall in the current maze blocking the view if you are say 6 from the end.

You can adapt the program to show your own maze but you must remember to change your conditions for 'EXIT'.

STRUCTURE	LINE(S)
DIMENSION ARRAYS AND FILL WITH DATA FOR MAZE MAP	100-110
INITIALIZE DIRECTION FACED AND FIRST LOCATION IN THE MAZE	130-136
THIS ROUTINE WILL DRAW THE RELEVANT LOCATION ON THE SCREEN BY CALLING SUBROUTINE AT LINE LOAD WHICH CALCULATES WHICH LINES SHOULD BE DRAWN BASED ON YOUR DISTANCE FROM THE END OF THE CORRIDOR YOU ARE IN	140-155
PRINTS THE EXIT SIGN IF YOU ARE NEAR THE EXIT	156
GETS THE PLAYERS INPUT AND CALCULATES NEXT POSITION IN CORRIDOR	160-185
TEST TO SEE IF PLAYER IS AT THE EXIT. GOTO END GAME IF TRIED	190
PROCESS INPUT COMMAND AND TAKE REQUIRED ACTION	200-310
THIS SUBROUTINE CALCULATES WHICH LINES TO DRAW BASED ON PLAYER'S POSITION IN MAZE	1000-1360
THESE ARE THE LINE DRAWING SUBROUTINES FOR THE PROGRAM	6000-8508
READS MAP DATA INTO THE VERTICAL AND HORIZONTAL ARRAYS	9005-9020
DATA FOR THE MAZE MAP '1' INDICATES A WALL '0' INDICATES A CORRIDOR	9100-9208

VARIABLES

UP CURSOR = MOVE FORWARD
LEFT CURSOR = TURN LEFT
RIGHT CURSOR = TURN RIGHT
DOWN CURSOR = TURN AROUND

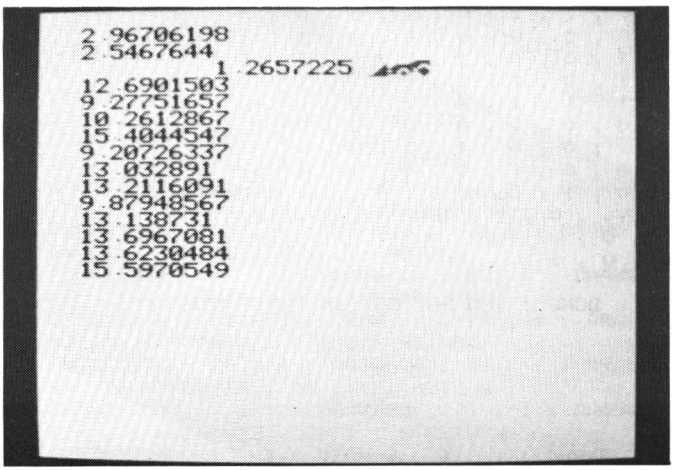
```
 40 REM * MAZEMAN *
 70 REM
100 DIM V(10 , 10) , H(10 , 10)
110 GOSUB 9000
120 X=1 : Y=1 : DX=1 : DY=0
125 ST=0
130 I=9 : IF DX=-1 THEN IX=X+1 ELSE IX=X
131 IF DY=-1 THEN IY=Y+1 ELSE IY=Y
132 I=I-1 : IX=IX+DX : IY=IY+DY
134 IF DX<>0 AND V(IX , IY)=0 THEN 132
136 IF DY<>0 AND H(IX , IY)=0 THEN 132
140 HIRES : PAPER5 : INK0 : GOSUB 6000+I
145 IF DX=-1 THEN IX=IX-1
146 IF DY=-1 THEN IY=IY-1
150 FOR K=I TO 8
155 IX=IX-DX : IY=IY-DY : GOSUB 1000 : NEXTK
156 IF DX=-1 AND Y=9 AND X<5 THEN CURSET  78 , 110 ,
    3 : GOSUB 9210
157 REM MAIN
158 GETA$
160 IF A$<>CHR$(11)THEN ST=ST+1 : GOTO 190
170 IF(DX =1 AND V(X+1 , Y)=0)THEN X=X+DX : Y=Y+DY :
    GOTO 190
175 IF(DX=-1 AND V(X , Y)=0)THEN X=X+DX : Y=Y+DY :
    GOTO 190
180 IF(DY=1 AND H(X , Y+1)=0)THEN X=X+DX : Y=Y+DY :
    GOTO 190
185 IF(DY=-1 AND H(X , Y)=0)THEN X=X+DX : Y=Y+DY
190 IF X=1 AND Y=9 THEN 9990
200 IF A$ =CHR$(10)THEN DX=-DX : DY=-DY : GOTO 130
210 IF A$=CHR$(8)THEN 300
215 IF A$ <>CHR$(9)THEN 130
220 IF ABS(DX)=1 THEN DY=-DX : DX=0 : GOTO 130
230 DX=DY : DY=0 : GOTO 130
300 IF ABS(DX)=1 THEN DY=DX : DX=0 : GOTO 130
310 DX=-DY : DY=0 : GOTO 130
1000 IF DX<>1 THEN 1100
1010 IF H(IX , IY+1)=0 THEN GOSUB 7500+K : GOTO 1050
1020 GOSUB 7000+K
1050 IF H(IX , IY)=0 THEN 8500+K
1060 GOTO 8000+K
1100 IF DX<>-1 THEN 1200
1110 IF H(IX , IY)=0 THEN GOSUB 7500+K : GOTO 1150
1120 GOSUB 7000+K
1150 IF H(IX , IY+1)=0 THEN 8500+K
1160 GOTO 8000+K
1200 IF DY<>-1 THEN 1300
1210 IF V(IX+1 , IY)=0THEN GOSUB 7500+K : GOTO 1250
1220 GOSUB 7000+K
1250 IF V(IX , IY)=0 THEN GOTO 8500+K
1260 GOTO 8000+K
1300 IF V(IX , IY)=0 THEN GOSUB 7500+K : GOTO 1350
1310 GOSUB 7000+K
```

```
1350 IF V(IX+1 , IY)=0 THEN 8500+K
1360 GOTO 8000+K
6000 CURSET110 , 76 , 1 : DRAW0 , 24 , 1 : DRAW18 ,
     0 , 1 : DRAW0 , -24 , 1 : DRAW-18 , 0 , 1 :
     RETURN
6001 CURSET108 , 75 , 1 : DRAW 0 , 26 , 1 : DRAW24 ,
     0 , 1 : DRAW0 , -26 , 1 : DRAW-24 , 0 , 1
     RETURN
6002 CURSET106 , 74 , 1 : DRAW0 , 28 , 1 : DRAW30 ,
     0 , 1 : DRAW0 , -28 , 1 : DRAW-30 , 0 , 1 :
     RETURN
6003 CURSET102 , 71 , 1 : DRAW0 , 34 , 1 : DRAW34 ,
     0 , 1 : DRAW0 , -34 , 1 : DRAW-34 , 0 , 1 :
     RETURN
6004 CURSET94 , 65 , 1 : DRAW0 , 46 , 1 : DRAW62 ,
     0 , 1 : DRAW0 , -46 , 1 : DRAW-62 , 0 , 1 :
     RETURN
6005 CURSET82 , 57 , 1 : DRAW0 , 62 , 1 : DRAW74 ,
     0 , 1 : DRAW0 , -62 , 1 : DRAW-74 , 0 , 1 :
     RETURN
6006 CURSET64 , 45 , 1 : DRAW0 , 86 , 1 : DRAW110 ,
     0 , 1 : DRAW0 , -86 , 1 : DRAW-110 , 0 , 1 :
     RETURN
6007 CURSET40 , 29 , 1 : DRAW0 , 118 , 1 : DRAW158 ,
     0 , 1 : DRAW0 , -118 , 1 : DRAW-158 , 0 , 1 :
     RETURN
6008 CURSET8 , 7 , 1 : DRAW0 , 162 , 1 : DRAW222 ,
     0 , 1 : DRAW0 , -162 , 1 : DRAW-222 , 0 , 1 :
     RETURN
7000 CURSET108 , 75 , 1 : DRAW2 , 1 , 1 : DRAW0 ,
     24 , 1 : DRAW-2 , 1 , 1 : RETURN
7001 CURSET106 , 74 , 1 : DRAW2 , 1 , 1 : DRAW0 ,
     26 , 1 : DRAW-2 , 1 , 1 : RETURN
7002 CURSET102 , 71 , 1 : DRAW4 , 3 , 1 : DRAW0 ,
     28 , 1 : DRAW-4 , 3 , 1 : RETURN
7003 CURSET94 , 65 , 1 : DRAW8 , 6 , 1 : DRAW0 , 34 ,
     1 : DRAW-8 , 6 , 1 : RETURN
7004 CURSET82 , 57 , 1 : DRAW12 , 8 , 1 : DRAW0 ,
     46 , 1 : DRAW-12 , 8 , 1 : RETURN
7005 CURSET64 , 45 , 1 : DRAW18 , 12 , 1 : DRAW0 ,
     62 , 1 : DRAW-18 , 12 , 1 : RETURN
7006 CURSET40 , 29 , 1 : DRAW24 , 16 , 1 : DRAW0 ,
     86 , 1 : DRAW-24 , 16 , 1 : RETURN
7007 CURSET8 , 7 , 1 : DRAW32 , 22 , 1 : DRAW0 ,
     118 , 1 : DRAW-32 , 22 , 1 : RETURN
7008 CURSET0 , 1 , 1 : DRAW8 , 6 , 1 : DRAW0 , 162 ,
     1 : DRAW-8 , 6 , 1 : RETURN
7500 CURSET108 , 76 , 1 : DRAW2 , 0 , 1 : DRAW0 ,
     24 , 1 : DRAW-2 , 0 , 1 : RETURN
7501 CURSET106 , 75 , 1 : DRAW2 , 0 , 1 : DRAW0 ,
     26 , 1 : DRAW-2 , 0 , 1 : RETURN
7502 CURSET102 , 74 , 1 : DRAW4 , 0 , 1 : DRAW0 ,
     28 , 1 : DRAW-4 , 0 , 1 : RETURN
7503 CURSET94 , 71 , 1 : DRAW8 , 0 , 1 : DRAW0 , 34 ,
     1 : DRAW-8 , 0 , 1 : RETURN
```

```
7504 CURSET82 , 65 , 1 : DRAW12 , 0 , 1 : DRAW0 ,
     46 , 1 : DRAW-12 , 0 , 1 : RETURN
7505 CURSET64 , 57 , 1 : DRAW18 , 0 , 1 : DRAW0 ,
     62 , 1 : DRAW-18 , 0 , 1 : RETURN
7506 CURSET40 , 45 , 1 : DRAW24 , 0 , 1 : DRAW0 ,
     86 , 1 : DRAW-24 , 0 , 1 : RETURN
7507 CURSET 8 , 29 , 1 : DRAW32 , 0 , 1 : DRAW0 ,
     118 , 1 : DRAW-32 , 0 , 1 : RETURN
7508 CURSET0 , 7 , 1 : DRAW8 , 0 , 1 : DRAW0 , 162 ,
     1 : DRAW-8 , 0 , 1 : RETURN
8000 CURSET130 , 75 , 1 : DRAW-2 , 1 , 1 : DRAW0 ,
     24 , 1 : DRAW2 , 1 , 1 : RETURN
8001 CURSET132 , 74 , 1 : DRAW-2 , 1 , 1 : DRAW0 ,
     26 , 1 : DRAW2 , 1 , 1 : RETURN
8002 C)RSET136 , 71 , 1 : DRAW-4 , 3 , 1 : DRAW0 ,
     28 , 1 : DRAW4 , 3 , 1 : RETURN
8003 CURSET144 , 65 , 1 : DRAW-8 , 6 , 1 : DRAW0 ,
     34 , 1 : DRAW8 , 6 , 1 : RETURN
8004 CURSET156 , 57 , 1 : DRAW-12 , 8 , 1 : DRAW0 ,
     46 , 1 : DRAW12 , 8 , 1 : RETURN
8005 CURSET174 , 45 , 1 : DRAW-18 , 12 , 1 : DRAW0 ,
     62 , 1 : DRAW18 , 0 , 1 : RETURN
8006 CURSET198 , 29 , 1 : DRAW-24 , 16 , 1 : DRAW0 ,
     86 , 1 : DRAW24 , 16 , 1 : RETURN
8007 CURSET230 , 7 , 1 : DRAW-32 , 22 , 1 : DRAW0 ,
     118 , 1 : DRAW32 , 22 , 1 : RETURN
8008 CURSET238 , 1 , 1 : DRAW-8 , 6 , 1 : DRAW0 ,
     162 , 1 : DRAW8 , 6 , 1 : RETURN
8500 CURSET130 , 76 , 1 : DRAW-2 , 0 , 1 : DRAW0 ,
     24 , 1 : DRAW2 , 0 , 1 : RETURN
8501 CURSET132 , 75 , 1 : DRAW-2 , 0 , 1 : DRAW0 ,
     26 , 1 : DRAW2 , 0 , 1 : RETURN
8502 CURSET136 , 74 , 1 : DRAW-4 , 0 , 1 : DRAW0 ,
     28 , 1 : DRAW4 , 0 , 1 : RETURN
8503 CUSET144 , 71 , 1 : DRAW-8 , 0 , 1 : DRAW0 ,
     34 , 1 : DRAW8 , 0 , 1 : RETURN
8504 CURSET156 , 65 , 1 : DRAW-12 , 0 , 1 : DRAW0 ,
     46 , 1 : DRAW12 , 0 , 1 : RETURN
8505 CURSET174 , 57 , 1 : DRAW-18 , 0 , 1 : DRAW0 ,
     62 , 1 : DRAW18 , 0 , 1 : RETURN
8506 CURSET198 , 45 , 1 : DRAW-24 , 0 , 1 : DRAW0 ,
     86 , 1 : DRAW24 , 0 , 1 : RETURN
8507 CURSET230 , 29 , 1 : DRAW-32 , 0 , 1 : DRAW0 ,
     118 , 1 : DRAW32 , 0 , 1 : RETURN
8508 CURSET238 , 7 , 1 : DRAW-7 , 0 , 1 : DRAW0 ,
     162 , 1 : DRAW7 , 0 , 1 : RETURN
9000 FOR I=1 TO 9 : H(I , 1)=1 : H(I , 10)=1 :
     FOR J=2 TO 9 : READ A : H(I , J)=A
9005 NEXT J : NEXT I
9010 FOR I=1 TO 9 : V(1 , I)=1 : V(10 , I)=1 :
     FOR J=2 TO 9 : READ A : V(J , I)=A
9015 NEXT J : NEXT I
9020 RETURN
9100 DATA 1 , 1 , 1 , 0 , 0 , 0 , 0 , 1
9101 DATA 0 , 0 , 1 , 0 , 0 , 1 , 0 , 1
9102 DATA 1 , 1 , 0 , 0 , 0 , 0 , 1 , 1
9103 DATA 0 , 0 , 1 , 1 , 0 , 0 , 1 , 0
```

```
9104 DATA 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0
9105 DATA 1 , 0 , 0 , 0 , 0 , 0 , 0 , 1
9106 DATA 1 , 1 , 1 , 0 , 1 , 0 , 0 , 1
9107 DATA 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0
9108 DATA 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0
9200 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
9201 DATA 1 , 1 , 1 , 1 , 0 , 1 , 0 , 0
9202 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1
9203 DATA 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0
9204 DATA 1 , 1 , 0 , 1 , 1 , 1 , 1 , 1
9205 DATA 1 , 1 , 1 , 1 , 1 , 1 , 0 , 1
9206 DATA 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0
9207 DATA 1 , 1 , 0 , 0 , 1 , 1 , 0 , 0
9208 DATA 0 , 0 , 0 , 1 , 0 , 0 , 0 , 1
9210 CHAR69 , 0 , 1 : CURMOV6 , 0 , 0 : CHAR88 , 0 ,
     1 : CURMOV6 , 0 , 0 : CHAR73 , 0 ,  :
     CURMOV6 , 0 , 0
9220 CHAR 84 , 0 , 1
9230 RETURN
9990 PRINT"WELL DONE!! YOU GOT OUT IN "ST " STEPS."
9995 END
```

Bubble Sort



There are many different sorts of sorts, but the sort this program deals with is the one that "arranges" things in a predetermined order.

In mathematics sort means to create order out of chaos, and the sorting process is usually applied to a group of numbers. In computing, sort means to take an "array" of numbers and sort them into numerical order or to take a group of "strings" and sort the characters or groups of characters into their alphabetical order.

Because characters of the alphabet have a numerical ASCII value in "Basic" and we sort them according to their value the sorting of strings is also a numerical sort.

The program given here serves both as a demonstration of the sorting process, as well as being the basis for a compact subroutine which you can use in your own programs - for example to sort a list of names and addresses or to determine if, in a game of cards, one card has a higher value than another or to find the highest number entered.

The sort demonstration program given here is a traditional bubble sort. The program compares two adjacent numbers in a list: if the numbers are the wrong way around (the larger number preceding the smaller number) then the computer swaps them around.

If you run the program you will soon see why it is called a bubble sort the smaller numbers seem to bubble up to the top as the larger numbers sink to take up the vacant spaces.

The program will continue to swap adjacent numbers, over and over again, each time getting the numbers sorted a little more until the sort is complete.

The program is very cute in its execution but it will soon be obvious that this method is also very slow.

MODIFIED BUBBLE SORT

The second program listing is a modified bubble sort. The program now roams much further afield to find a number to swap with. Doing it this way it effectively cuts the number of swaps down to about half of those in the above, traditional, bubble sort.

You can enter the second program as listed or if you have already entered the first program, all you need to do is change lines 390, 400, 410, 600 and 610.

SORT SUBROUTINES

Although this is a sorting demonstration program it is obvious that the purpose of this program is to provide you with a sorting routine that you can use in your own programs.

Program 3 gives the listing of the modified bubble sort without graphics and with the sorting process contained within a subroutine (lines 500 to 530) which you can use in your own programs. This subroutine assumes that the numbers to be sorted are in an array P(n) and that n is the number of elements to be sorted.

As you can see from running this program, 15 numbers can be sorted in about 3 seconds including the time taken to print the 15 numbers out twice. More time still, can be saved if the PRINTs were not included and this would be the case in your own programs.

Sort 1

```
100 REM * SORT-1 *
110 REM
120 LB$="            " : B$=LB$+"      "
130 D$=CHR$(1)+"[\]^"+CHR$(6)
140 PAPER6 : INK0 : CLS
150 RESTORE
160 POKE 61 , PEEK(618)AND254
170 FOR I=46808 TO 46839 : READ V : POKE I , V :
    NEXT I
180 N=15
190 DIM P(N)
200 PRINT"Do you want to choose your own"
210 PRINT"numbers (y or n)?"
220 K$=KEY$ : IF LEN(K$)=0 THEN 220
230 IF K$="n" OR K$="N" THEN 310
240 PRINT"Enter 15 numbers to be sorted"
250 POKE 618 , PEEK(618)OR1
260 FOR Y=1 TO N
270 GOSUB 680 : P(Y)=A
280 NEXT Y
290 POKE 618 , PEEK(618)AND254
300 GOTO 340
310 FOR Y=1 TO N
320 P(Y)=15*RND(1)+1
330 NEXT Y
340 CLS
350 FOR Y=1 TO N
360 PLOT1 , Y , CHR$(0)+MID$(STR$(P(Y)) , 2)
370 NEXT Y
380 M=18
390 FOR I=N-1 TO 1 STEP -1
400 FOR J=1 TO I
410 L=J+1
420 IF P(J)<=P(L) THEN 600
430 PJ$=MID$(STR$(P(J)) , 2) :
    PL$=CHR$(0)+MID$(STR$(P(L)) , 2)
440 IF L=M THEN 460
450 FORY=M TO L STEP SGN(L-M) : PLOT18 , Y , D$ :
    PLOT18 , Y+SGN(M-L) , "    ↑   " : NEXTY
460 A=LEN(PL$)+1 : FORX=17TOASTEP-1 : PLOTX , L ,
    D$+" " : NEXTX
470 FOR X=1TO13 : PLOTX , L , " "+PL$+D$ : NEXT
480 FORY=LTOJSTEP-1 : PLOT13 , Y , PL$+D$ : PLOT13 ,
    Y+1 , B$ : NEXTY
490 PLOT1 , J , CHR$(3)+PJ$
500 FOR Y=J+1TOL : PLOT1 , Y-1 , LB$ : PLOT1 , Y ,
    CHR$(3)+PJ$
510 IFY<>J+1THENPLOT1 , Y-1 ,
    CHR$(0)+MID$(STR$(P(Y-1)) , 2)
520 NEXTY
530 PLOT1 , L , CHR$(0)+PJ$
540 FOR X=13TO1STEP-1 : PLOTX , J , PL$+D$+" " :
    NEXTX
```

12# — 6# — 6#

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

```
550 FOR X=AT018 : PLOTX , J , " "+D$ : NEXTX
560 M=J
570 T=P(L)
580 P(L)=P(J)
590 P(J)=T
600 NEXT J
610 NEXT I
620 FORY=MT018 : PLOT18 , Y , D$ : PLOT18 , Y-1 ,
    "   " : NEXTY
630 Y=21 : X=1 : GOSUB 730 : PRINT"Do you want to sort
    again? "
640 K$=KEY$ : IF LEN(K$)=0 THEN 640
650 IF K$="Y" OR K$="y" THEN RUN
660 POKE 618 , PEEK(618)OR1
670 END
680 INPUT A$ : A=VAL(A$) : IF A=0 AND LEFT$(A$ ,
    1)="0" OR A<>0 THEN 720
690 SHOOT
700 PRINT"ERROR! - You must key in a number!"
710 GOTO 680
720 RETURN
730 POKE 616 , Y : DOKE 18 , Y*40+48000
740 POKE 61 , X
750 RETURN
760 V$=CHR$(SC)+MID$(STR$(V) , 2) : RETURN
770 DATA 0 , 0 , 0 , 1 , 3 , 7 , 15 , 63
780 DATA 0 , 0 , 35 , 39 , 63 , 38 , 62 , 32
790 DATA 0 , 3 , 63 , 63 , 6 , 19 , 56 , 16
800 DATA 63 , 63 , 49 , 36 , 30 , 31 , 14 , 4
```

6#

Sort2

```
100 REM * SORT-2 *
110 REM
120 LB$="            " : B$=LB$+"      "
130 D$=CHR$(1)+"[\]^"+CHR$(6)
140 PAPER6 : INK0 : CLS
150 RESTORE
160 POKE 618 , PEEK(618)AND254
170 FOR I=46808 TO 46839 : READ V : POKE I , V : NEXT I
180 N=15
190 DIM P(N)
200 PRINT"Do you want to choose your own"
210 PRINT"numbers (y or n)?"
220 K$=KEY$ : IF LEN(K$)=0 THEN 220
230 IF K$="n" OR K$="N" THEN 310
240 PRINT"Enter 15 numbers to be sorted"
250 POKE 618 , PEEK(618)OR1
260 FOR Y=1 TO N
270 GOSUB 680 : P(Y)=A :
280 NEXT Y
290 POKE 618 , PEEK(618)AND254
300 GOTO 340
310 FOR Y=1 TO N
320 P(Y)=15*RND(1)+1
330 NEXT Y
340 CLS
350 FOR Y=1 TO N
360 PLOT1 , Y , CHR$(0)+MID$(STR$(P(Y)) , 2)
370 NEXT Y
380 M=18
390 FOR J=1 TO N-1
400 FOR I=J+1 TO N
410 L=N+J-I+1
420 IF P(J)<=P(L) THEN 600
430 PJ$=MID$(STR$(P(J)) , 2) :
    PL$=CHR$(0)+MID$(STR$(P(L)) , 2)
440 IF L=M THEN 460
450 FORY=M TO L STEP SGN(L-M) : PLOT18 , Y , D$ :
    PLOT18 , Y+SGN(M-L) , "      " : NEXTY
460 A=LEN(PL$)+1 : FORX=17TOASTEP-1 : PLOTX , L ,
    D$+" " : NEXTX
470 FOR X=1TO13 : PLOTX , L , " "+PL$+D$ : NEXT
480 FORY=LTOJSTEP-1 : PLOT13 , Y , PL$+D$ : PLOT13 ,
    Y+1 , B$ : NEXTY
490 PLOT1 , J , CHR$(3)+PJ$
500 FOR Y=J+1TOL : PLOT1 , Y-1 , LB$ : PLOT1 , Y ,
    CHR$(3)+PJ$
510 IFY<>J+1THENPLOT1 , Y-1 ,
    CHR$(0)+MID$(STR$(P(Y-1)) , 2)
520 NEXTY
530 PLOT1 , L , CHR$(0)+PJ$
540 FOR X=13TO1STEP-1 : PLOTX , J , PL$+D$+" " :
    NEXTX
550 FOR X=ATO18 : PLOTX , J , " "+D$ : NEXTX
```

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

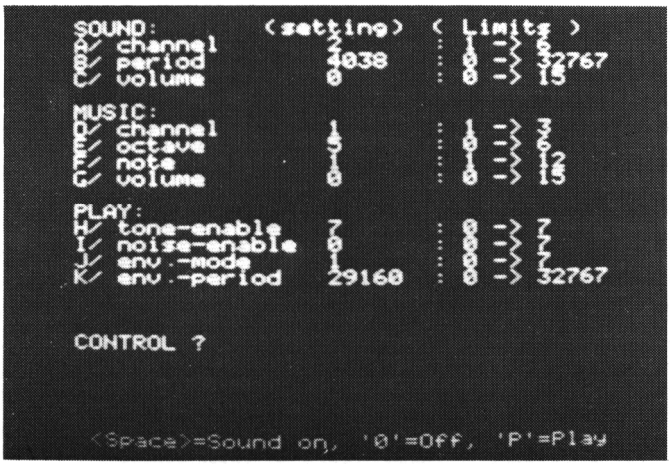
```
560 M=J
570 T=P(L)
580 P(L)=P(J)
590 P(J)=T
600 NEXT I
610 NEXT J
620 FORY=MTO18 : PLOT18 , Y , D$ : PLOT18 , Y-1 ,
    "  ↑    " : NEXTY
630 Y=21 : X=1 : GOSUB 730 : PRINT"Do you want to
    sort again? "
640 K$=KEY$ : IF LEN(K$)=0 THEN 640
650 IF K$="Y" OR K$="y" THEN RUN
660 POKE 618 , PEEK(618)OR1
670 END
680 INPUT A$ : A=VAL(A$) : IF A=0 AND
    LEFT$(A$ , 1)="0" OR A<>0 THEN 720
690 SHOOT
700 PRINT"ERROR! - You must key in a number!"
710 GOTO 680
720 RETURN
730 POKE 616 , Y : DOKE 18 , Y*40+48000
740 POKE 617 , X
750 RETURN
760 V$=CHR$(SC)+MID$(STR$(V) , 2) : RETURN
770 DATA 0 , 0 , 0 , 1 , 3 , 7 , 15 , 63
780 DATA 0 , 0 , 35 , 39 , 63 , 38 , 62 , 32
790 DATA 0 , 3 , 63 , 63 , 6 , 1 , 56 , 16
800 DATA 63 , 63 , 49 , 36 , 30 , 31 , 14 , 4
```

6#

Sort3

```
100 REM * SORT-3 *
110 PAPER6 : INK0 : CLS
120 POKE 618 , PEEK(618)AND254
130 N=15
140 DIM P(N)
150 PRINT"Do you want to choose your own"
160 PRINT"numbers (y or n)?"
170 K$=KEY$ : IF LEN(K$)=0 THEN 170
180 IF K$="n" OR K$="N" THEN 260
190 PRINT"Enter 15 numbers to be sorted"
200 POKE 618 , PEEK(618)OR1
210 FOR Y=1 TO N
220 GOSUB 360 : P(Y)=A :
230 NEXT Y
240 POKE 618 , PEEK(618)AND254
250 GOTO 290
260 FOR Y=1 TO N
270 P(Y)=15*RND(1)+1
280 NEXT Y
290 GOSUB 440 : GOSUB490
300 GOSUB 440
310 Y=21 : X=1 : GOSUB 410 : PRINT"Do you want to
    sort again? "
320 K$=KEY$ : IF LEN(K$)=0 THEN 320
330 IF K$="Y" OR K$="y" THEN RUN
340 POKE 618 , PEEK(618)OR1
350 END
360 INPUT A$ : A=VAL(A$) : IF A=0 AND LEFT$(A$ ,
    1)="0" OR A<>0 THEN 400
370 SHOOT
380 PRINT"ERROR! - You must key in a number!"
390 GOTO 360
400 RETURN
410 POKE 616 , Y : DOKE 18 , Y*40+48000
420 POKE 617 , X
430 RETURN
440 CLS
450 FOR Y=1TON
460 PRINTP(Y)
470 NEXT
480 RETURN
490 REM *** SORT ROUTINE
500 FOR J=1 TO N-1 : FOR I=J+1 TO N : L=N+J-I+1
510 IF P(L)<P(J)THENM=J : T=P(L) : P(L)=P(J) :
    P(J)=T
520 NEXTI , J
530 RETURN
```

Sound Aid/Synthesizer



The program starts off with a demonstration consisting of a few set notes (stored in a Data Statement) played with random patches.

After the program leaves the demonstration it enters the control mode which gives the user complete control of the 8912 sound chip.

The controls are listed onto the screen along with their current settings (depending on the last setting from the demonstration program). To 'patch' a sound, simply enter the letter alongside the desired control followed by your choice of setting - hit RETURN.

If the sound is set on then you will hear the sound change when you hit RETURN.

Once you find a sound you like then you can enter play mode which allows you to play tunes using the top three rows of the ORIC keyboard. You can return to control mode at any time.

HOW THE PROGRAM WORKS

The demonstration note sequence is stored as data and can be changed to a string of data of any length and any note sequence (i.e. a tune).

The position of each control on the screen is stored in an array to simplify the display of patch changes.

To achieve instantaneous output from each keypress during play mode, a method of storing the note values into an array and accessing the array by the ASCII code of the key pressed is used.

This method would be useful in programs such as games that use several keys for control.

The method used in 'Sound Aid' is as follows :

1) Set a counter to zero

2) Increment counter

3) Read next keyboard control character from data

4) Use ASC() Statement to calculate character's ASCII value

5) Use this value as a pointer to the element in the array being used

6) Store the value of the counter into this element of the array

7) If more data then go to Step (2)

8) Array is now ready to use

USING THE ARRAY

There are two methods in which this array can now be used to an advantage. One method is to test the array inside an ON GOTO statement e.g. ON k(n) GOTO 100,200,300...etc.

Or as used in the 'Sound Aid' program, the actual value stored in each element of the array is used as the input for the sound commands.

Example:
The first 12 keys of each of the top three rows of the keyboard are used as the Synthesiser keyboard. If we wanted to output the THIRD semitone of the SECOND octave we would want to input to the MUSIC command :

MUSIC channel, 2, 3, volume.

We would hit the third key across on the second-top row. The program would get this character, (the letter 'D'), calculate its ASCII value with the ASC() command (which would be 68) and use this value as the array element pointer. The program would then extract the contents of this array and find the value 15 (i.e. 1 x 12 + 3) which was loaded into the array during initialization. The octave is calculated by taking the integer value of (15/12) which is 1 (the first octave). The note is calculated by subtracting the (octave x12) from 15 which is 15 - 1x12 = 3 (the third note).

This method saves the use of continual scanning of 36 letters of the keyboard which would cause about a 2 second delay between notes.

STRUCTURE	LINE(s)
Initialization	100-195
Display	200-280
Patch control	300-395
Sound output	500-580
Play mode	2000-2095
Sound demonstration	700-790
Data	1000-1130

VARIABLES

```
C$() = Control name
L() = Lower limit of each control
U() = Upper limit of each control
V() = Current value of each control
N() = Array for demo music
R() = Controls that will be given random values
      during demo
K() = Keyboard array for 'Play' mode
A$ = Keyboard character during 'Play' mode
OC = Octave during 'Play' mode
N$ = Keyboard character during 'Control' mode
V = Setting during 'Control' mode
P = Display pointer for editing controls
```

```
100 REM * SOUND AID *
120 POKE 618 , 10  : REM CURSOR & CLICK OFF
130 INK 6  : PAPER 0
140 DIM C$(20) , L(20) , U(20) , V(20) , N(50) ,
    R(20) , K(93)
150 Y=0  : E=11 : FLG=0
160 FOR K=1 TO E
170 READ C$(K) , L(K) , U(K)
180 NEXT K
190 FOR I=1 TO E  : READ V(I)  : NEXT
200 FOR K=1 TO 11 : READ R(K) : NEXT : I=1
210 READN(I) : IFN(I)<>0THENI=I+1 : GOTO210 ELSE
    NE=I-1
220 FOR K=1 TO 36 : READ D$ : K(ASC(D$))=K : NEXT :
    K(13)=12
230 CLS : PRINT"         *** SOUND DEMO ***" : PRINT :
    PRINT"HIT ANY KEY FOR SYNTHESISER"
240 GOSUB 660
250 REM
260 CLS
270 FOR K=1 TO Y : PRINT : NEXT K
280 PRINT"SOUND :      (setting)   ( Limits )"
290 FOR K=1 TO E
300 IF K=4 THEN PRINT : PRINT"MUSIC : "
310 IF K=8 THEN PRINT : PRINT"PLAY : "
320 IF K=12 THEN PRINT
330 PRINTCHR$(K+64);"/ ";C$(K);
340 POKE617 , 19 : PRINT INT(V(K));
350 POKE617 , 26 : PRINT" :  ";L(K);"-> ";U(K)
360 NEXT K
370 REM
380 PAPER 0
390 PLOT 1 , 26 , CHR$(1)+"<Space>=Sound on ,
    'O'=Off ,  'P'=Play"
400 FOR K=20 TO 25
410 POKE616 , K :  PRINT : POKE617 , 1 : PRINTSPC(37); :
    NEXT K
420 POKE616 , 21 : PRINT : POKE617 , 1 :
    PRINTSPC(37);
430 POKE616 , 20 : PRINT : POKE617 , 1
440 PRINT"CONTROL ?  ";
450 N$=KEY$ : IF N$="" THEN 450
460 IF N$="O"THEN FLG=0 : PLAY0 , 0 , 0 , 0
470 IF N$="P"THEN FLG=1 : GOSUB610
480 IFN$="P"THEN 920
490 IF N$<"A" OR N$>"K"THEN 400
500 N=ASC(N$)-64 : PRINT N$;" = ";V(N) : PRINT
510 INPUT"SETTING";V
520 IF V<L(N)ORV>U(N)THEN400
530 V(N)=V : P=N : IF N>3 THEN P=P+2
540 IF N>7 THEN P=P+2
550 IF N>11THEN P=P+1
560 P=P+Y+1
```

8#

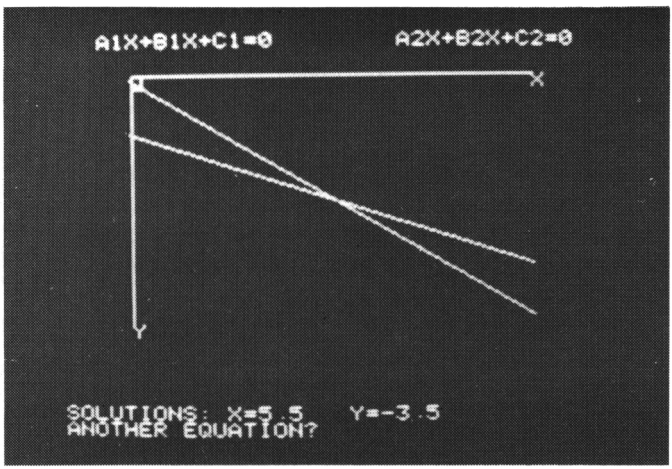
Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: 12#.

```
570 PLOT 17 , P , STR$(V(N))+"    "
580 IF FLG=0THEN600
590 GOSUB610 : GOTO 600
600 IF PEEK(616)>24 THEN 250 ELSE 400
610 REM * PLAY IT *
620 SOUND V(1) , V(2) , V(3)
630 MUSIC V(4) , V(5) , V(6) , V(7)
640 PLAY  V(8) , V(9) , V(10) , V(11)
650 RETURN
660 REM * DEMO *
670 FORK=1TONE : V(6)=N(K) : GOSUB610 : WAIT20 :
    PLAY0 , 0 , 0 , 0 : NEXT
680 IF KEY$<>""THEN RETURN
690 FOR K=1 TO 8  : N=R(K)
700 R=INT(RND(1)*U(N))
710 IF R<L(N) THEN R=R+1
720 V(N)=R
730 NEXT
740 IF RND(1)<.8THENV(8)=7 : V(9)=0 : V(10)=1
750 GOTO 660
760 REM
770 DATA channel , 1 , 6 , period , 0 , 32767 ,
    volume , 0 , 15
780 DATA channel , 1 , 3 , octave , 0 , 6 , note , 1
    , 12 , volume , 0 , 15
790 DATA tone-enable , 0 , 7 , noise-enable , 0 , 7 ,
    env.-mode , 0 , 7
800 DATA env.-period , 0 , 32767
810 REM * INITIAL VALUES *
820 DATA 1 , 100 , 0 , 1 , 3 , 1 , 0 , 1 , 0 , 4 ,
    100
830 REM RANDOMIZABLES
840 DATA 1 , 2 , 4 , 5 , 8 , 9 , 10 , 11 , 12 , 3 , 7
850 REM * DEMO MUSIC DATA *
860 DATA 1 , 5 , 8 , 1
870 DATA 0
880 REM * KEYBOARD DATA *
884 REM * THIS IS THE KEYBOARD DATA THAT IS LOADED *
886 REM * INTO THE ARRAY K IN LINE 220 *
890 DATA A , S , D , F , G , H , J , K , L , ";" ,
    ":" , ?
900 DATA Q , W , E , R , T , Y , U , I , O , P ,
    [ , ]
910 DATA 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 0 ,
    - , =
920 REM * PLAY MODE *
930 PAPER 4
940 FOR K=20 TO 26 : POKE616 , K : PRINT : POKE617 ,
    1 : PRINTSPC(37); : NEXT K
950 PLOT 1 , 20 , "       *** P L A Y - M O D E ***"
960 PLOT 1 , 22 , "Use top 3 rows to play"
970 PLOT 1 , 24 , "Hit <Space> to return."
980 PLAY 0 , 0 , 0 , 0 : GET A$ :  IF A$=" "THEN370
    ELSE P=PEEK(520)
990 IF A$>"]"THEN 980
1000 IF A$=""THEN K=K(39) ELSE K=K(ASC(A$))
```

6#

```
1010 IF K=0 THEN 980
1020 OC=INT((K-1)/12)+1
1030 MUSIC V(4) , OC , K-(OC-1)*12 , V(7)
1040 PLAY V(8) , V(9) , V(10) , V(11)
1050 IF PEEK(520)=P THEN 1050 ELSE 980
```

Simultaneous Equations



This program solves two simultaneous equations of the type :

ax + by + c = o

This equation is the general form of equation for a straight line and the 'solution' of two such lines is the mathematical term for the point where the two lines intersect.

You will be asked by the program to enter the values of a, b and c for each of the two equations, and the program will find out for what values of x and y the lines meet.

If there is no solution - in other words the lines are parallel and do not meet - the program will tell you so.

The program will then draw for you each of the two lines and show where they cross on a relative scale.

HOW THE PROGRAM WORKS :

The actual calculation part of the program is fairly small - most of the program consists of getting the correct information from you, and then displaying the results to you both numerically and graphically (by actually drawing the graph).

The determinant D is calculated in lines 820-840. If the determinant is zero, then the program tells you that there is no solution i.e. the lines are parallel and do not meet.

Lines 850-1990 determine in which quadrant the intersection of the two lines occur, and in which direction the graphs should be plotted.

The PLOT command in the TEXT, LORES 0 and LORES 1 modes does not actually move the cursor. Usually, there is no need to do so except when you wish to input information. A useful subroutine which will allow the user to move the cursor anywhere on the screen is located at line 10. Memory locations 616 and 617 contain the y and x coordinates of the cursor on the display. Poking different values into these memory locations will move the cursor. P$ is CHR$(17), which turns the cursor off and on to make the display more presentable.

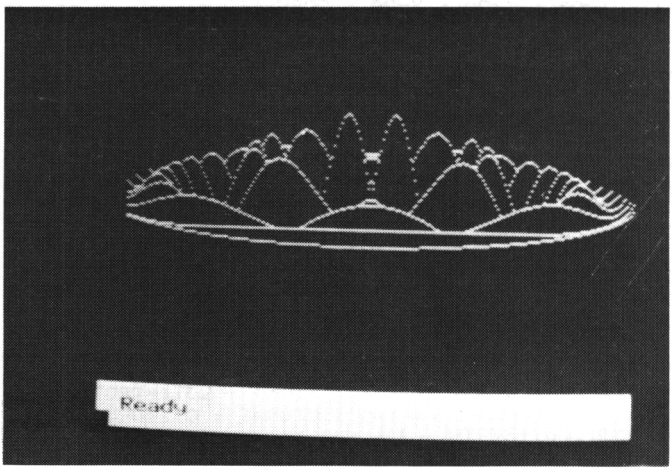
Another useful subroutine is located at line 30, which will draw the string of characters contained in I$, anywhere on the HIRES screen.

STRUCTURE	LINE(s)
Subroutine which first turns off the cursor, moves the cursor "X" across the screen and "Y" down the screen. The cursor is then reactivated.	110
Subroutine which draws in the HIRES mode the O, X and Y labels, which are abbreviations for Origin, X-axis and Y-axis.	130-150
Subroutine which draws the string of characters contained in I$ on the HIRES screen.	160
Subroutine which rubs out the string of characters contained in I$ on the HIRES screen.	170
Sets up screen in TEXT mode.	180-220
Initializes all arrays.	230
Inputs both first and second equation.	240-250
Works out determinant D. If no solution exists i.e. D=O, then no solution exists.	260-280
Works out and displays solution numerically.	290-310
Determines in which quadrant the intersection of the two lines occurs and then draws the K-Y axis.	320-400
Asks the user if he wishes to rerun the program. If not, then the program ends.	420-450
Works out and plots the line of the first equation on an adjusted scale to show the intersection of the two lines.	460-600
Plots the line of the second equation.	610-750

```
100 GOTO180
110 POKE616 , Y : PRINTP$ : POKE617 , X : PRINTP$;
    : RETURN
130 FORI=1TO3 : CURSETVAL(MID$(I$ , I*2-1 , 1))*40 ,
    VAL(MID$(I$ , I*2 , 1))*40 , 0
140 CHARASC(MID$("OXY" , I , 1)) , 0 , 1 : NEXT
150 OX=VAL(MID$(I$ , 7 , 3)) : OY=VAL(MID$(I$ , 10)) :
    RETURN
160 FORI=1TOLEN(I$) : CURSETX+6*I , Y , 0 :
    CHARASC(MID$(I$ , I , 1)) , 0 , 1 : NEXT :
    RETURN
170 FORI=1TOLEN(I$) : CURSETX+6*I , Y , 0 :
    CHARASC(MID$(I$ , I , 1)) , 0 , 0 : NEXT :
    RETURN
180 TEXT : CLS : P$=CHR$(17) : INK2 : PAPER0
190 PLOT9 , 0 , "SIMULTANEOUS EQUATIONS"
200 PLOT1 , 2 , "a1X+b1Y+c1=0" : PLOT20 , 2 ,
    "a2X+b2Y+c2=0
210 FORI=1TO2 : I$=RIGHT$(STR$(I) , 1)+" =" :
    PLOT(I-1)*19+1 , 4 , "a"+I$
220 PLOT(I-1)*19+1 , 5 , "b"+I$ : PLOT(I-1)*19+1 ,
    6 , "c"+I$ : NEXT
230 DIMC(6) , S$(7) , X$(7) , Y$(7)
240 FORT=1TO2 : FORI=1TO3 : Y=I+3 : X=(T-1)*19+7 :
    GOSUB110 : INPUTD : C(I+(T-1)*3)=D
250 GOSUB110 : PRINTSPC(14) : GOSUB110 : PRINTD :
    NEXTI , T
260 X=1 : Y=8 : GOSUB110
270 D=C(2)*C(4)-C(1)*C(5)
280 IFD=0THENPRINT"NO SOLUTIONS!!" : WAIT100 : RUN
290 A=(C(3)*C(5)-C(2)*C(6))/D
300 B=(C(1)*C(6)-C(3)*C(4))/D
310 PRINT"SOLUTIONS :  X="A"  Y="B
320 PRINT : INPUT"DO YOU WANT A PLOT (Y/N)";I$ :
    IFLEFT$(I$ , 1)="N"THENPRINT : GOTO420
330 HIRES
340 IFA>=0ANDB>=0THENI$="14541140 160" : GOSUB130
350 IFA>=0ANDB<0THENI$="11511440 40" : GOSUB130
360 IFA<0ANDB>=0THENI$="541451200160" : GOSUB130
370 IFA<0ANDB<0THENI$="51115420040" : GOSUB130
380 DX=SGN(A)-(A=0)
390 DY=(SGN(B)>=0)-(SGN(B)<0)
400 CURSETOX , OY , 1 : DRAWDX*160 , 0 , 1 :
    CURSETOX , OY , 1 : DRAW0 , DY*120 , 1
410 GOSUB460
420 INPUT"ANOTHER EQUATION";K$ : K$=LEFT$(K$ , 1)
430 IFK$="N"THENTEXT : END
440 IFK$<>"Y"THEN420
450 RUN
460 RX=ABS(80/A) : RY=ABS(60/B)
470 IFC(1)=0THENXA=0 : YA=C(3)/C(2) : XB=2*A :
    YB=YA : GOTO550
480 IFC(2)=0THENYA=0 : XA=-C(3)/C(1) : YB=-2*B :
    XB=XA : GOTO550
490 XA=2*A : YA=(C(3)+C(1)*XA)/C(2)
500 IFDY*YA<0THENYA=0 : GOTO520
```

```
510 IF(ABS(YA)-ABS(2*B))>0THENYA=-2*B
520 XB=0 : YB=C(3)/C(2)
530 IFDY*YB<0THENYB=0 : GOTO550
540 IF(ABS(YB)-ABS(2*B))>0THENYB=-2*B
550 CURSETOX+RX*XA , OY+RY*YA , 1
560 DRAW(XB-XA)*RX , (YB-YA)*RY , 1
570 I$="A1X+B1X+C1=0" : X=20 : Y=20 : GOSUB160 :
    WAIT100
580 GOSUB170
590 CURSETOX+RX*XA , OY+RY*YA , 0
600 DRAW(XB-XA*RX , (YB-YA)*RY , 0
610 IFC(4)=0THENXC=0 : YC=C(6)/C(5) : XD=2*A :
    YD=YC : GOTO690
620 IFC(5)=0THENYC=0 : XC=-C(6)/C(4) : YD=-2*B :
    XD=XC : GOTO690
630 XC=2*A : YC=(C(6)+C(4)*XC)/C(5)
640 IFDY*YC<0THENYC=0 : GOTO660
650 IF(ABS(YC)-ABS(2*B))>0THENYC=-2*B
660 XD=0 : YD=C(6)/C(5)
670 IFDY*YD<0THENYD=0 : GOTO690
680 IF(ABS(YD)-ABS(2*B))>0THENYD=-2*B
690 CURSETOX+RX*XC , OY+RY*YC , 1
700 DRAW(XD-XC)*RX , (YD-YC)*RY , 1
710 I$="A2X+B2X+C2=0" : X=140 : Y=20 : GOSUB160 :
    WAIT100
720 CURSETOX+RX*XA , OY+RY*YA , 1
730 DRAW(XB-XA)*RX , (YB-YA)*RY , 1
740 PRINT"SOLUTIONS :  X="A"  Y="B
750 I$="A1X+B1X+C1=0" : X=20 : Y=20 : GOSUB160 :
    RETURN
```

3D Plot



This program plots a three dimensional "picture" using formulae (functions). The function it uses is found in line 180.

If we take an ellipse (fig 1) and draw say 5 chords across it, we can then choose points along each chord. These points having a distance from the origin (R) and plot a point (P) where (P) is always a function of (R).

By doing this along each chord we can create the illusion of mountains in 3 dimensions. By keeping track of the highest point plotted we can ensure that we do not plot any points which lie "behind" this point thus creating a truly 3 dimensional effect.

You can use any function you like as long as the function is always a function of (r).

For example you could try:

1. `F = (R - 1) * Cos (8 * R)`
2. `F = (R - 1) * Exp (R/2)`
3. `F = (1 - R) * Exp (Cos (12 * R))`

To run this game just load it and "RUN" it.

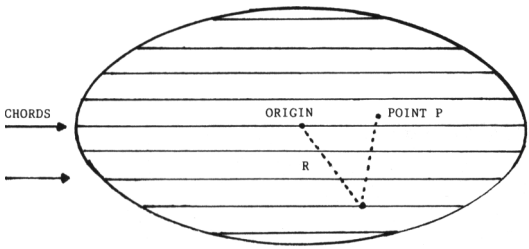
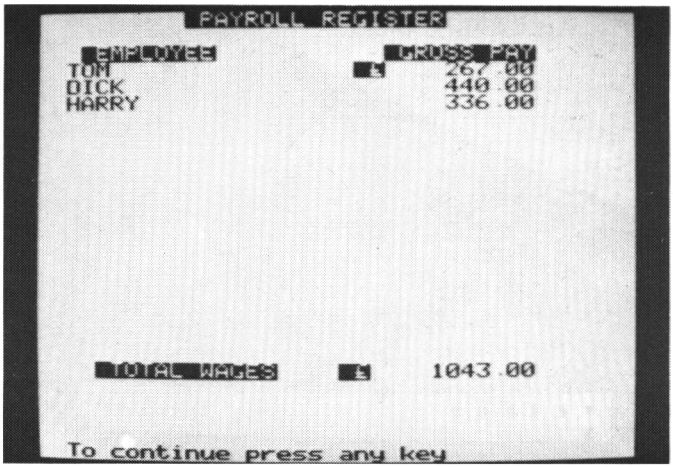


FIG. 1

```
 10 REM * * * * * * * * * * *
 20 REM * 3D PLOT *
 30 REM * * * * * * * * * * *
100 HIRES:PAPER 0:INK 7:H=239:V=199
110 REM FUNCTION IN LINE 180
120 X1=H/2 : X2=X1*X1 : Y1=V/2 : Y2=V/4
130 FOR X=0 TO X1
140 X4=X*X : L=-Y1
150 A=SQR(X2-X4) : REM CHORD LEN
160 FOR I=-A TO A STEP V/10
170 R=SQR(X4+I*I)/X1
180 F=(1-R)*SIN(18*R)
190 Y=I/5+F*Y2
200 IF Y<=L THEN 230
210 L=Y : Y=Y1-Y
220 CURSETX1-X , Y , 1 : CURSETX1+X , Y , 1
230 NEXT I : NEXT X
240 END
```

Payroll



This program calculates gross wages of up to 18 employees, and is therefore suitable for small businesses.

PAYROLL begins by offering the option of entering new information on employees - name and hourly rate and OT factor or restoring employee details previously saved on tape.

After this information is either entered or restored the user is prompted to enter each employee's hours - both regular and overtime. Gross wages for each employee and the total wages bill are calculated and displayed on the next screen.

The program is user-friendly, prompting for all input and informing the user when data tapes need to be loaded.

To the user the program appears as three screens:
1. EMPLOYEE DETAILS
This has two modes - one for entering new information, the other for displaying previously entered information.
2. HOURLY PAYROLL REGISTER
This has the same two modes as EMPLOYEE DETAILS.
3. WAGES PAYROLL REGISTER
This has two modes calculate/display and display. The screens are cycled through in the order given above by any keypress. After each cycle of 3 screens, the user is given the option of exiting from the program.

```
PROGRAM STRUCTURE

INITIALIZATION: 1000-1130
Initialize mode of three screens to enter/calculate.
Initialize screen to ENTER/RESTORE option.

EMPLOYEE DETAILS: 1140-1600
If ENTER option chosen:
- input number of employees (1-18)
- define following arrays: E$ = Employee details
                           R$ = Regular hours
                           O$ = Overtime hours
                           G$ = Gross income

ED: If mode is ENTER
- set mode to DISPLAY
- input employees names and hourly rate
- input overtime factor
- save EMPLOYEE DETAILS option offered
- return

If RESTORE option chosen:
- input EMPLOYEE DETAILS from tape.
Due to limitations of ORIC, the program must be run again at this point.
It will, however, continue from this point.
- define arrays: E$, R$, O$, G$.
Fill E$ with restored data, set mode to DISPLAY.


If mode is DISPLAY
- display employee details
- return

PAYROLL HOURS REGISTER 2000-2620
Set up headings:
If mode is ENTER
- set mode to DISPLAY
- input regular and overtime hours
- return

If mode is DISPLAY
- display hours register
- return

PAYROLL GROSS INCOME REGISTER 4000-4210
If mode is CALCULATE
- set mode to DISPLAY
- initialize total gross wages
- calculate each employee's wages and total wages
- enter these into corresponding arrays
Set up headings
If mode is DISPLAY
- display wages report
- return
```

```
MAIN LOOP 9000-9060
EMPLOYEE DETAILS
L1: PAYROLL HOURS REGISTER
PAYROLL WAGES REGISTER
GOSUB ED
GOTO L1
```

NOTES

As ORIC does not support data files per se, saving and loading data is done by writing the contents of E$ to a known block of memory, and saving this block; loading the block and writing to E$ from the block.

Structure of E$
- first record:
  1-5: number of employee records in the array
  16-21: overtime factor
- following records
  1-15: employee name
  16-21: hourly rate

```
 100 REM PAYROLL
 110 REM mini payroll program
 115 X=PEEK(#1CA5) : IF X=1 THEN RM=1 : GOTO 5000
 120 RM=0 : PM=0 : PR=0 : REM set enter mode
 130 PAPER 7 : INK 0 : CLS : GOTO 9000
1000 REM
1010 REM initialize routine
1020 PLOT 6 , 0 , CHR$(7)+CHR$(16)+"EMPLOYEE
     INFORMATION"+CHR$(23)+CHR$(0)
1030 PLOT 1 , 6 , "ENTER new information ---- E"
1040 PLOT 1 , 8 , "RETRIEVE information ---- R"
1050 PLOT 8 , 15 , CHR$(12)+"E"+CHR$(8)+"or"
     +CHR$(12)+"R"+CHR$(8)
1060 REPEAT : GET K$ : UNTIL K$="E" OR K$="R" OR
     K$="e" OR K$="r"
1070 IF K$="e" OR K$="r" THEN K$=CHR$(ASC(K$)-32)
1080 PLOT 8 , 15 , CHR$(8) : PLOT 13 , 15 , CHR$(8)
1090 IF K$<>"R" GOTO 1150
1100 PLOT 10 , 18 , CHR$(7)+CHR$(16)+"RESTORE"+
     CHR$(23)+CHR$(0) : RM=1
1110 PLOT 1 , 22 , "Press <space> when ready"
1120 GET K$ : IF K$ <> " " GOTO 1105
1130 POKE #1CA5 , 1 : CLOAD "DATA" , A#1B00 , E#1CA5
1140 F=VAL(E$(1 , 15)+E$(1 , 16)+E$(1 , 17)+E$
     (1 , 18)+E$(1 , 19)+E$(1 , 20)+E$(1 , 21))
1145 GOTO 1190
1150 POKE 616 , 18 : PRINT : POKE 617 , 1
1155 INPUT "ENTER no. of employees (1-18)";NE
1160 IF NE>18 OR NE<0 THEN PLOT 31 , 18 , "    " :
     GOTO 1150
1180 DIM E$(NE+1 , 21)
1190 DIM R$(NE , 6)
1200 DIM O$(NE , 6)
1210 DIM G$(NE , 8)
1220 WAIT 25
1230 CLS : PLOT 8 , 0 , CHR$(7)+CHR$(16)+"EMPLOYEE
     DETAILS"+CHR$(23)+CHR$(0)
1240 PLOT 2 , 2 , CHR$(7)+CHR$(16)+"NAME OF
     EMPLOYEE"+CHR$(23)+CHR$(0)
1245 PLOT 21 , 2 , CHR$(7)+CHR$(16)+"HOURLY
     RATE"+CHR$(23)+CHR$(0)
1250 IF RM=1 GOTO 1500
1260 REM enter employee details
1270 RM=1
1280 FOR I=2 TO NE+1
1290 FOR J=1 TO 15 : E$(I , J)=" " : NEXT J
1295 FOR J=16 TO 21 : E$(I , J)="0" : NEXT J
1300 LI=I+1
1325 POKE 616 , LI : PRINT : POKE 617 , 2
1330 INPUTS$ : IF S$="*" THEN RETURN
1335 T$=S$
1340 FOR J=1 TO LEN(S$) : E$(I , J)=LEFT$(T$ , 1) :
     T$=MID$(T$ , 2 , LEN(T$)-1) : NEXT J
```

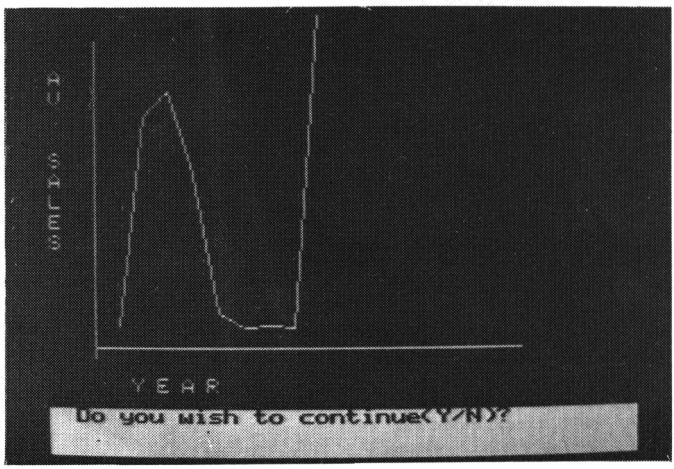
```
1345 PLOT 1 , LI , " "
1350 POKE616 , LI : PRINT : POKE 617 , 21
1360 INPUT S$ : IF VAL(S$)<0 GOTO 1350
1370 NS=3 : GOSUB 3000 : REM input numeric string
1380 PLOT 20 , LI , "  "+S$+"   "
1385 T$=S$
1390 FOR J=1 TO LEN(S$) : E$(I , J+15)=MID$
     (S$ , J , 1) : NEXT J
1395 NEXT I
1400 PLOT 9 , 21 , CHR$(7)+CHR$(16)+"Overtime
     factor"+CHR$(23)+CHR$(0)
1405 POKE 616 , 21 : PRINT : POKE 617 , 29
1410 INPUT S$ : IF VAL(S$)<0 GOTO 1405
1415 GOSUB 3000
1420 F=VAL(S$) : PLOT 28 , 21 , "  "+S$+"  "
1425 FOR J=1 TO LEN(S$) : E$(1 , J+15)=MID$
     (S$ , J , 1) : NEXT J
1430 T$=STR$(NE) : T$=MID$(T$ , 2) : IF LEN(T$)<2
     THEN T$=" "+T$
1435 E$(1 , 1)=LEFT$(T$ , 1) :
     E$(1 , 2)=RIGHT$(T$ , 1)
1440 WAIT 50
1445 POKE 616 , 23 : PRINT : POKE 617 , 1
1450 INPUT "Save EMPLOYEE DATA (Y/N)";K$
1460 IF K$="Y" OR K$="y" THEN GOSUB 6000 : RETURN
1470 IF K$<>"N"AND K$<>"n" THEN PLOT 26 ,
     23 , "   " : GOTO 1445
1480 RETURN
1500 REM redisplay employee record
1510 NE=VAL(E$(1 , 1)+E$(1 , 2))
1515 F=VAL(E$(1 , 16)+E$(1 , 17)+E$(1 , 18)+E$
     (1 , 19)+E$(1 , 20)+E$(1 , 21))
1520 FOR I=2 TO NE+1
1530 LI=I+1 : T$="" : S$=""
1535 FOR J=1 TO 15 : T$=T$+E$(I , J) : NEXT J
1540 FOR J=16 TO 21 : S$=S$+E$(I , J) : NEXT J
1545 PLOT 2 , LI , T$ : PLOT 23 , LI , S$
1550 NEXT I
1560 PLOT 9 , 21 , CHR$(7)+CHR$(16)+"OVERTIME
     FACTOR"+CHR$(23)+CHR$(0)
1570 PLOT 28 , 21 , MID$(STR$(F) , 2)
1580 PLOT 1 , 23 , "Press any key to continue"
1590 GET K$
1600 RETURN
2000 REM
2010 REM enter/display payroll hour
2020 CLS : PLOT 8 , 0 , CHR$(7)+CHR$(16)+"PAYROLL
     REGISTER"+CHR$(23)+CHR$(0)
2030 PLOT 17 , 1 , CHR$(7)+CHR$(16)+"---- HOURS
     ----"+CHR$(23)+CHR$(0)
2040 PLOT 1 , 2 , CHR$(7)+CHR$(16)+"EMPLOYEE"
     +CHR$(23)+CHR$(0)
2045 PLOT 19 , 2 , CHR$(7)+CHR$(16)+"REG"+CHR$(23)
     +CHR$(0)
2047 PLOT 27 , 2 , CHR$(7)+CHR$(16)+"OT"+CHR$(23)
     +CHR$(0)
```

```
2050 IF PM=1 GOTO 2500
2060 PM=1 : TR=0 : TD=0
2070 FOR I=2 TO NE+1
2080 LI=I+1 : T$=""
2085 FOR J=1 TO 15 : T$=T$+E$(I , J) : NEXT J
2090 PLOT 1 , LI , T$
2100 POKE 616 , LI : PRINT : POKE 617 , 21
2110 INPUT N1 : IF N1<0 GOTO 2100
2120 S$=MID$(STR$(N1) , 2) : NS=3 : GOSUB 3000
2130 PLOT 17 , LI , " "+S$  : TR=TR+N1
2135 FOR J=1 TO LEN(S$) : R$(I-1 , J)=MID$
     (S$ , J , 1) : NEXT J
2140 POKE 616 , LI : PRINT : POKE 617 , 27
2150 INPUT N1 : IF N1<0 GOTO2140
2160 S$=MID$(STR$(N1) , 2) : NS=3 : GOSUB 3000
2170 PLOT 24 , LI , "  "+S$
2180 TD=TD+N1
2185 FOR J=1 TO LEN(S$) : O$(I-1 , J)=MID$
     (S$ , J , 1) : NEXT J
2190 NEXT I
2200 S$=MID$(STR$(TR) , 2) : NS=4 : GOSUB 3000
2210 PLOT 5 , 21 , CHR$(7)+CHR$(16)+"TOTALS"
     +CHR$(23)+CHR$(0)
2215 PLOT 17 , 21 , S$
2220 S$=MID$(STR$(TD) , 2) : GOSUB 3000
2230 PLOT 25 , 21 , S$
2235 PLOT 1 , 26 , "Press any key to continue" :
     GET K$
2240 RETURN
2500 REM display hour report
2510 FOR I=1 TO NE
2520 LI=I+2
2525 T$="" : FOR J=1 TO 15 : T$=T$+E$(I+1 , J) :
     NEXT J
2530 PLOT 1 , LI , T$
2533 T$="" : FOR J=1 TO 6 : T$=T$+R$(I , J) : NEXT J
2535 PLOT 20 , LI , T$
2537 T$="" : FOR J=1 TO6 : T$=T$+O$(I , J) : NEXT J
2540 PLOT 28 , LI , T$
2545 NEXT I
2550 PLOT 8 , 21 , CHR$(7)+CHR$(16)+"TOTALS"
     +CHR$(23)+CHR$(0)
2560 S$=MID$(STR$(TR) , 2) : NS=4 : GOSUB3000
2561 END
2570 PLOT 20 , 21 , S$
2580 S$=MID$(STR$(TD) , 2) : GOSUB 3000
2590 PLOT 28 , 21 , S$
2600 PLOT 1 , 23 , "Press any key to continue"
2610 GET K$
2620 RETURN
3000 REM
3010 REM string input of value
3020 T$=MID$(STR$(INT(VAL(S$))) , 2)
3025 U$=MID$(S$ , LEN(T$)+2)
3030 IF LEN(T$)<NS THEN T$=" "+T$ : GOTO3030
3035 IF LEN(T$)>NS THEN T$=RIGHT$(T$ , NS)
```

```
3045 T$=T$+"."
3050 IF VAL(U$)=0 GOTO 3090
3060 FOR J=1 TO LEN(U$) : IF MID$(U$ , J , 1)="."
     GOTO 3080
3070 NEXT J
3080 U$=LEFT$(U$ , J-1)
3090 IF LEN(U$)>2 THEN U$=LEFT$(U$ , 2)
3095 IF LEN(U$)<2 THEN U$=U$+"0" : GOTO 3095
3100 S$=T$+U$
3110 RETURN
4000 REM
4010 REM calculate/display pay
4020 IF PR=1 GOTO 4100
4030 PR=1 : TP=0
4040 FOR I=1 TO NE
4043 T1$="" : FOR J=1 TO 6 : T1$=T1$+R$(I , J) :
     NEXT J
4045 T2$="" : FOR J=1 TO 6 : T2$=T2$+O$(I , J) :
     NEXT J
4047 T3$="" : K=I+1 : FOR J=16 TO 21 :
     T3$=T3$+E$(K , J) : NEXT J
4050 N1=(VAL(T1$)+VAL(T2$)*F)*(VAL(T3$))
4060 S$=MID$(STR$(N1) , 2) : NS=5 : GOSUB3000
4061 S$=S$+"    "
4070 FOR J=1 TO 8 : G$(I , J)=MID$(S$ , J , 1) :
     NEXT J
4080 TP=TP+N1
4090 NEXT I
4100 CLS : PLOT8 , 0 , CHR$(7)+CHR$(16)+"PAYROLL
     REGISTER"+CHR$(23)+CHR$(0)
4110 PLOT 1 , 2 , CHR$(7)+CHR$(16)+"EMPLOYEE"
     +CHR$(23)+CHR$(0)
4115 PLOT 21 , 2 , CHR$(7)+CHR$(16)+"GROSS
     PAY"+CHR$(23)+CHR$(0)
4117 PLOT 19 , 3 , CHR$(7)+CHR$(16)+"$"+CHR$(23)
     +CHR$(0)
4120 FOR I=1 TO NE
4130 LI=I+2
4140 S$="" : FOR  J=1 TO 15 : S$=S$+E$(I+1 , J) :
     NEXT J
4143 PLOT 1 , LI , S$
4145 S$="" : FOR J=1 TO 8 : S$=S$+G$(I , J) : NEXT J
4147 PLOT 24 , LI , S$
4150 NEXT I
4160 PLOT 2 , 21 , CHR$(7)+CHR$(16)+"TOTAL
     WAGES"+CHR$(23)+CHR$(0)
4165 PLOT 18 , 21 ,
     CHR$(7)+CHR$(16)+"$"+CHR$(23)+CHR$(0)
4170 S$=MID$(STR$(TP) , 2) : NS=6 : GOSUB 3000
4180 PLOT 23 , 21 , S$
4190 PLOT 1 , 26 , "To continue press any key"
4200 GET K$
4210 RETURN
5000 REM load employee data
5030 T1$=CHR$(PEEK(#1B00+2))
5040 T2$=CHR$(PEEK(#1B00+3))
```

```
5050 NE=VAL(T1$+T2$)
5060 DIM E$(NE+1 , 21)
5070 FOR I=1 TO NE+1
5080 FOR J=1 TO 21
5090 E$(I , J)=CHR$(PEEK(#1B00+I+J))
5100 NEXT J : NEXT I
5110 GOTO 1140
6000 REM save employee data
6020 FOR I=1 TO NE+1
6030 FOR J=1 TO 21
6033 IF LEN(E$(I , J))=0 THEN T$=" " ELSE T$=E$
    (I , J)
6040 POKE #1B00+I+J , ASC(T$)
6050 NEXT J : NEXT I
6055 CLS : PLOT 5 , 5 , "Insert data tape"
6057 PLOT 5 , 8 , "Press <space> when ready" :
     GET K$
6060 CSAVE "DATA" , A#1B00 , E#1CA5
6070 RETURN
9000 REM
9010 REM main routine
9020 GOSUB 1000 : REM initialize
9030 GOSUB 2000 : REM payroll register
9040 GOSUB 4000 : REM display/calc pay
9050 GOSUB 1230 : REM payroll record
9060 CLS : PLOT 5 , 5 , "To exit press <space>" :
     PLOT 7 , 6 , "Any other key to continue"
9070 GET K$ : IF K$<>" " GOTO 9030
9080 CLS : END
```

Sales Analysis



This program uses quarterly sales histories of a company to calculate seasonal indices. These indices can then be used to predict sales and sales trends for that company.

This will benefit sales managers in industries where sales are affected by seasonal fluctuations. For example the sales of ski equipment will be highest during the autumn and winter periods.

After the data has been entered, the program will calculate seasonal ratios by dividing the actual sales by the average quarterly sales for all years entered.

The average of each quarter's ratios over the years produces the seasonal index.

After the graph has been plotted a question is asked: "Do you wish to continue (Y/N)?". By answering Y the user has the option to restart with a new set of data. This is particularly useful if the user wants to compare different sets of data and observe the trends of each set of data.

PROGRAM NOTE

The program as it is written will only take £9999 as the maximum amount per quarter. Any company or business earning more than this per quarter has to enter the amount in multiples of ten.

Since the amount of money we are dealing with is so large, only whole pounds should be entered. If you enter pounds and pence then the program will round off to the nearest whole pound.

The program can only take sales analysis for 17 years inclusive, the reason for this is the number of lines available to print each sales figure is limited to 17.

A graph is produced at the end showing the average sales each year against the year. Naturally the graph would be more accurate and show a true indication of sales trends if many years' data are entered.

HOW THE PROGRAM WORKS

The program jumps to a main loop, where there is a series of "gosubs" which contain the following routines:

```
MAIN-LOOP
Input year range;
Input sales history;
Calculator;
Plot-graph;
Print "Do you wish to continue (Y/N)";
If Yes, then goto main-loop.
```

INPUT-YEAR-RANGE
Asks two questions, which tell the program from which year to which year the program should calculate the sales analysis. If these questions are incorrectly entered then a prompt which exists until the data is valid.

INPUT-SALES-HISTORY
Display the year for which the data is to be entered.
A prompt is printed for each quarter.
After the four quarters have been entered then the average is displayed for WAIT (500).
A loop is done until all the years have been entered.
At the end the table is a display with the year, each quarter and the average sales.
A question is asked if the user wants this table to be printed out.

CALCULATES
This routine calculates the indices for each quarter by dividing the sales for that quarter by the average sales for that year.
A final calculation is done - the quarterly average is then taken and displayed at the bottom as the index for each quarter.

PLOT-GRAPH
Plots the graph in hi-res mode.
The lowest average sales is in variable L; and the highest average sales in variable H. The following calculation is made:
Ratio = Int ((H-L)/170)
To get a point on the graph then the following is done:

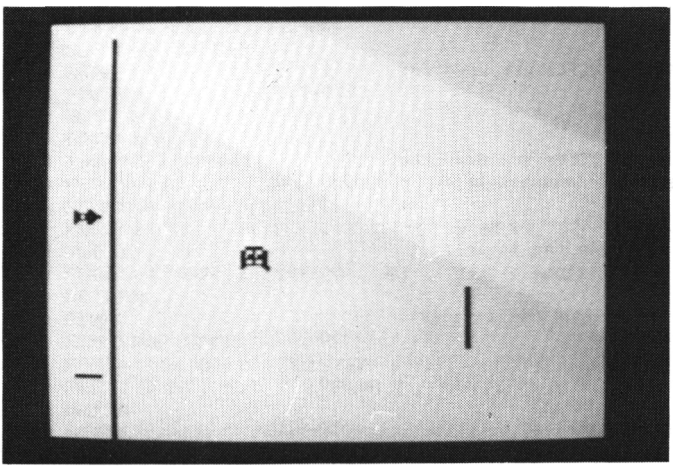
```
X = 0
For I = 1 to NY
Y = A(I)*Ratio   ; Y co-ordinate
X = X + 10       ; X co-ordinate
plot X, Y
Next I
```

```
100 REM * SALES ANALYSIS *
110 REMSEASONAL INDEX
120 PAPER3 : CLS : INK4 : GOTO1250
130 REM**************************
140 REM INPUT  YEAR  RANGE    *
150 REM**************************
160 PLOT 11 , 1 , CHR$(1)+CHR$(23)+"SEASONAL
    INDEX "+CHR$(19)
170 PRINT : PRINT : PRINT : PRINT
180 PRINT" SALES HISTORY (17 years inclusive)"
190 X=5 : Y=10 : GOSUB1220 : INPUT"From YEAR";S$
200 Y1=INT(VAL(S$))
210 IF Y1<1900 OR LEN(S$)>4 THEN190
220 X=5 : Y=12 : GOSUB1220 : INPUT"To   YEAR";S$
230 Y2=INT(VAL(S$))
240 IF Y2<1900 OR LEN(S$)>4 OR Y2<Y1 OR Y2>Y1+17
    THEN 220
250 NY=Y2-Y1+1
260 DIMY$(NY , 4)  'YEAR
270 DIMH(NY , 4)'SALES HISTORY
280 DIMA(NY)  'ANNUAL AVR SALES
290 DIMR(NY , 4)'RATIOS
300 DIMD$(4)  'SUM OF QUARTER RATIO
310 FORI=Y1 TO Y2 : Y$(I-Y1+1 , 4)=STR$(I) : NEXTI
320 RETURN
330 REM
340 REM*****************************
350 REM   INPUT SALES HISTORY
360 REM*****************************
370 IF HM=1 THEN630
380 FORI=1 TO NY
390 CLS : PLOT11 , 1 , CHR$(1)+CHR$(23)+"SALES
    HISTORY"+CHR$(19) : PRINT
400 X=0 : Y=4 : GOSUB1220
410 PRINT"Please enter average sales" : PRINT :
    PRINT"for YEAR ";Y$(I , 4) : PRINT
420 TS=0 : X=0 : Y=6 : FORJ=1 TO 4 : Y=Y+3
430 GOSUB1220
440 PRINT"Quarter " , J : INPUTS$
450 IFLEN(S$)>4ORLEN(S$)<1THEN430
460 IFVAL(S$)=0THEN430
470 N1=INT(VAL(S$)) : TS=TS+N1
480 H(I , J)=N1
490 NEXTJ
500 S=TS/4 : E=0.005 : EX=100 : GOSUB1190
510 PRINT : PRINT : PRINT"Average sale for ";
    Y$(NY , 4);" is ";S
520 A(I)=S
530 NEXTI
540 CLS : PLOT11 , 1 , CHR$(1)+CHR$(23)+"SALES
    HISTORY"+CHR$(19) : PRINT
550 X=0 : Y=4 : GOSUB1220
560 PRINT"YEAR  QTR1 QTR2 QTR3 QTR4   AV. SALES"
570 PRINT : Y=4
```

```
580 FORI=1 TO NY : X=-4
590 FORK=1TO4 : G$(K)=STR$(H(I , K)) : NEXTK
600 X=X+5 : PLOTX , Y , Y$(I , 4)
610 FORK=1TO4 : X=X+5 : PLOTX , Y , G$(K) : NEXTK
620 X=X+9 : PLOTX , Y , STR$(A(I)) : Y=Y+1 : NEXTI
630 REM
640 REM DISPLAY
650 GOSUB1140
660 RETURN
670 REM CALCULATES
680 CLS
690 E=0.00005 : EX=1000
700 FORI=1 TO NY
710 PING
720 FORJ=1 TO 4
730 S=H(I , J)/A(I) : GOSUB1190 : R(I , J)=S
740 NEXTJ : NEXTI
750 CLS : PLOT9 , 1 , CHR$(1)+CHR$(23)+"COMPUTED
    RATIOS"+CHR$(19)
760 X=1 : Y=5 : GOSUB1220
770 PRINT"YEAR    QTR1   QTR2   QTR3   QTR4"
780 Y=5 : FORI=1TONY : X=-4
790 FORK=1TO4 : G$(K)=STR$(R(I , K)) : NEXTK : X=X+5
    : PLOTX , Y , Y$(I , 4)
800 FORK=1TO4 : X=X+7 : PLOTX , Y , G$(K) : NEXTK
810 L$=LEFT$(G$(K) , LEN(G$(K)))
820 IFL$="."THENPLOTX+1 , Y , G$(K)ELSEPLOTX , Y ,
    G$(K)
830 Y=Y+1 : NEXTI
840 FORI=1TO4 : T=0
850 FORJ=1TONY
860 T=T+R(J , I)
870 NEXTJ
880 D$(I)=STR$(T)
890 NEXTI : X=8
900 PLOT1 , 23 , "INDEX" : FORI=1TO4 : PLOTX , 23 ,
    D$(I) : X=X+7 : NEXTI
910 GOSUB1140
920 RETURN
930 REM
940 REM **** PLOT GRAPH *****
950 REM
960 CLS : HIRES
970 CURSET20 , 170 , 3 : DRAW170 , 0 , 1
980 CURSET20 , 20 , 3 : DRAW0 , 155 , 1
990 CURSET35 , 185 , 3
1000 N$="YEAR" : FORI=1TOLEN(N$) : CHARASC(MID$(N$ ,
    I , 1)) , 0 , 1
1010 CURSET35+(I*10) , 185 , 3 : NEXTI
1020 CURSET1 , 35 , 3 : N$="AV. SALES" :
    FORI=1TOLEN(N$) : CHARASC(MID$(N$ , I , 1) ,
    0 , 1
1030 CURMOVE0 , 10 , 0 : NEXTI
1040 H=0 : L=A(1)
1050 FORI=1TONY
1060 IFH<A(I)THENH=A(I)
```

```
1070 IFL>A(I)THENL=A(I)
1080 NEXTI : R=(H-L)/150
1081 IF R=0THEN R=H/150
1090 CURSET30 , 160 , 3 : Y=L/R
1100 FORI=1TONY : Y=Y-INT(A(I)/R) : DRAW10 , Y , 1
1110 Y=INT(A(I)/R)
1120 NEXTI
1130 RETURN
1140 REM
1150 X=0 : Y=26 : GOSUB1220
1160 INPUT"  To continue hit Y";S$
1170 RETURN
1180 GOTO1160
1190 REM NUMBER TRIMMING
1200 S=INT((S+E)*EX)*(1/EX)
1210 RETURN
1220 A1=PEEK(616)*40+PEEK(617)+48000 : POKE616 , Y :
     DOKE18 , Y*40+48000
1230 POKE617 , X : POKEA1 , PEEK(A1)AND127
1240 RETURN
1250 TEXT
1260 REM MAIN LOOP
1270 HM=0 : CM=HM : GM=HM
1280 GOSUB130 : REM INPUT YEAR RANGE
1290 GOSUB330 : REM INPUT SALES HISTORY
1300 GOSUB670 : REM CALCULATES
1310 GOSUB930 : REM PLOT GRAPH
1320 INPUT"Do you wish to continue(Y/N)";S$
1330 IFS$="Y"THEN RUN
```

Space Escape



The object of Space Escape is to move the invader from its starting position to the flashing star on the far right hand of the screen.

The player must contend with a firing spaceship and the force of gravity which will continually drag the player's character back to its original starting position.

Controls :

The 'cursor up' key will move the player's character upwards towards the top of the screen.

The 'cursor right' key will move the player's character to the right (towards the end point, the flashing star).

However it must be noted that to gain sufficient motion towards the right, one must have sufficient height - i.e. the higher the player's position is the further the player may move to the right. Also if no keys are being pressed the player's character will be pulled "by the force of gravity" back to its original starting position.

STRUCTURE

FUNCTION	LINE(s)
Turn cursor and keyclick off	120
Defines user defined graphics and reads data at 9000	150-210
These strings (TB$ and SP$) are used for the "laser beam"	220-230
Draw Tower routine - Using the defined character set draw the tower the invader positions itself at	240-280
Trap back routine - This for next loop pulls the invader back to the tower position	290-340
This for next loop returns the invader to the cell position	350-390
Main loop of program	410
Initialize and set variables	450-470
These two formulae, using boolean tests, determine if the UP (cursor up) key or Forward key (cursor right) are pressed, and thus the co-ordinates of the player's character (H & D).	480-510
Plots the character movement on the screen	560-630
If the positions of Ship and Player are equal then "fire".	710
Fire Routine : For next loop fires laser beam. Decides if player piece has been hit	710
Performs trap back routine if player has been hit	770
Message Routine : This simple routine prints a message according to whether or not the player has escaped or been hit by the spaceship	780-860

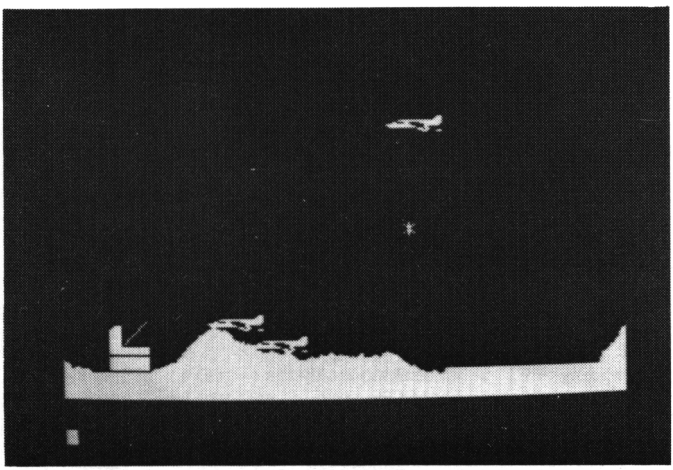
VARIABLES

H = Height : Y co-ordinate
D = Distance : X co-ordinate
SH=FR= Fire Flag
TIME = Time to finish game
O9 = Position of spacecraft

```
100 PAPER6 : INK0 : CLS
110 REM * SPACE ESCAPE *
120 POKE 618 , 10
130 GOTO410
140 REM DEFINE SHIP SHAPE
150 READ D$ : IF D$="END" THEN 220
160 B=#B400+ASC(D$)*8
170 FOR P=B TO B+7
180 READ D
190 POKE P , D
200 NEXT P
210 GOTO 150
220 TB$="" : SP$=""
230 FOR I=1TO 27 : SP$=SP$+" " : TB$=TB$+"-" : NEXT
240 REM *** DRAW TOWER
250 FORT=1TO26 : PLOT3 , T , "1" : NEXT
260 PLOT1 , 22 , "kk1"
270 FORB1=17TO20 : PLOT28 , B1 , "1" : NEXT
280 RETURN
290 REM
300 REM TRAP BACK
310 FORI=(ID+(D>ID))TO1STEP-1
320 PLOTI , H , "ab " : PLOTI , H+1 , "cd "
330 WAIT4
340 NEXTI
350 FORJ= HTO24
360 PLOT1 , J , "  " : PLOT1 , J+1 , "  "
370 PLOT1 , J+1 , "ab" : PLOT1 , J+2 , "cd"
380 WAIT4
390 NEXTJ
400 RETURN
410 REM
420 REM MAIN LOOP
430 RESTORE : GOSUB140 : REM DEFINE USER GRAPHIC FOR
    SHIP
440 GOSUB 240 : REM DRAW TOWER
450 S=30 : H=20 : D=5 : SH=0 : OG=INT(RND(1)*17)+2 :
    FR=SH
460 TIME=0
470 OH=H : OD=D
480 LETK$=KEY$ : IF K$=""THEN K=0 ELSE K=ASC(K$)
490 H=H-(H<19)+(K=11ANDH>1)*1.5 : IH=INT(H)
500 TIME=TIME+1
510 D=D-(K=9ANDH<19)*2+(D >5)*.3 : ID=INT(D)
520 IFK<>9THEN PLAY0 , 0 , 0 , 0
530 IFD>37THEND=37
540 IFSH=0THEN600
550 SH=0
560 PLOT30 , 18 , " "
570 PLOTOD , OH , "  " : PLOTOD , OH+1 , "  "
580 PLOTD , H , "ef" : PLOTD , H+1 , "gh"
590 GOTO640
600 SH=1
```

```
610 PLOTOD , OH , "  " : PLOTOD , OH+1 , "  "
620 PLOTD , H , "ab" : PLOTD , H+1 , "cd"
630 PLOT3 , 18 , "*" : WAIT2 :
640 IFD=>27THENP=21 : GOTO810
650 IFFR=0ANDRND(1)<.9 ORH>19THEN470
660 IFFR=1THEN710
670 NG=INT(INT(RND(1)*2)+H)-1 : NG=NG-(G>18)
680 IFNG<2THENNG=2
690 GM=(NG<OG)-(NG>OG)
700 FR=1
710 IFNG=OGTHEN740
720 PLOT1 , OG , "  " : OG=OG+GM : PLOT1 , OG , "ij"
730 GOTO470
740 FORL=5TOOD-2 : PLOTL , NG , "-" : NEXT : ZAP
750 PLOT5 , NG , SP$
760 IFNG<>IH AND NG<>IH+1OR H>23THENFR=0 : GOTO470
770 GOSUB290
780 T$=RIGHT$(STR$(TIME) , LEN(STR$(TIME))-1)
790 PLOT2 , 21 , "YOU LASTED "+T$+" CENTONS"
800 WAIT200 : GOTO840
810 PLOT2 , 21 , "YOU ESCAPED!"
820 WAIT200
830 REM TIMER
840 IFKEY$="W"ORKEY$="D"THEN840
850 CLS : GOTO440
860 IFKEY$=""THEN860
870 REM *** DATA FOR SHAPES
880 DATA a , 0 , 0 , 0 , 30 , 30 , 49 , 61 , 61
890 DATA b , 0 , 0 , 0 , 6 , 60 , 198 , 246 , 246
900 DATA c , 63 , 49 , 49 , 63 , 14 , 14 , 2 , 0
910 DATA d , 254 , 198 , 198 , 254 , 56 , 6 , 32 , 0
920 DATAe , 0 , 0 , 0 , 15 , 15 , 49 , 49 , 55
930 DATAf , 0 , 0 , 0 , 248 , 248 , 198 , 198 , 222
940 DATAg , 55 , 63 , 49 , 49 , 63 , 112 , 96 , 64
950 DATAh , 222 , 254 , 198 , 198 , 254 , 7 , 3 , 1
960 DATAi , 32 , 48 , 63 , 57 , 57 , 63 ,  48 , 32
970 DATAj , 240 , 248 , 252 , 63 , 63 , 252 , 248 ,
    240
980 DATAk , 0 , 0 , 0 , 0  , 0  , 0 , 63 , 63
990 DATAl , 3 , 3 , 3 , 3 , 3 , 3 , 3 , 3
1000 DATAEND
```

Bomber



The main aim of this game is to survive an onslaught of 20 Bombers. This isn't easy as your missiles may only damage the bombers or you might hit one and it then proceeds to crash land right on top of you. Therefore you should take care when firing at them ensuring that you hit them properly and far away enough to make a "safe" kill. Your controls are straight forward; "up arrow" for turret up and "right arrow" for turret down. The "space bar" is used to fire with. Because you are using an A.A. flak gun your projectiles explode on reaching a certain height (this to ensure that a near miss still causes damage). You have a set of controls with which to adjust your delay time (time taken to explode after firing). They are "left arrow" to increase delay time and "down arrow" to decrease delay time. Using these keys you can explode the missile at the height of the bomber.

In this program we use HIRES mode, as this permits us to set the turret angles. If we were to use LORES, we would have to redefine 34 characters to represent the 17 turret positions. The characters used from the standard character set, to draw the bombers, are ASCI 33-41. DRAW is used to draw the landscape on the bottom of the screen. This command has also been used to define the gun's turret at its different settings.

STRUCTURE	LINE(S)
SET UP AND DEFINE CHARS	0-200
DRAW LANDSCAPE	200-1000
EXPLOSION ROUTINE	1000-2000
MAIN LOOP	2000-2920
CHECK KEYBOARD FOR ACTION	2000-2070
INCREASE FLAK DELAY	2100
SET UP BOMBER	2110-2150
MOVE BOMBER	2170-2310
SET UP A BOMB	2320-2360
MOVE BOMB	2380-2460
MOVE MISSILE 1	2500-2670
MOVE MISSILE 2	2700-2870
CHECK BOMBER FOR HIT	2880-2920
ELEVATE GUN	3000-3040
SET UP MISSILE 1	3080-3090
SET UP MISSILE 2	3100-3110
END OF GAME MESSAGE	4000-4130

VARIABLES

S(), C() = CO-ORDINATES FOR GUN BARREL
NP = NUMBER OF PLANES
A = ANGLE OF GUN
TB = FALLING BOMB TIME
TP = FALLING PLANE TIME
TM, TN = MISSILE FLIGHT TIME
TL = FLAK TIME
XM, YM, XN, YN = MISSILE CO-ORDINATES
XB, YB = BOMB COORDINATES
XP, YP = PLANE COORDINATES
KP, KM = SCALING CONSTANTS

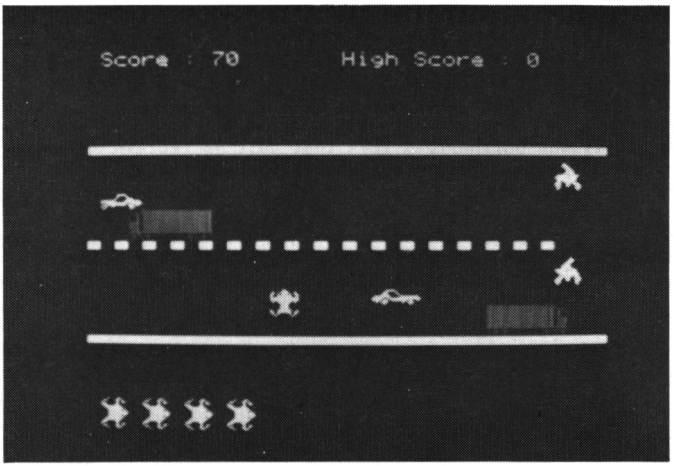
```
 10 REM ***  B O M B E R  ***
 11 REM
 20 POKE 618 , 10
 30 DIMS(13) , C(13)
 40 PAPER4 : INK0 : HIRES : CLS :
    PRINT"PLEASE WAIT......";
 50 FORA=0TO13 : READS(A) : READC(A) : NEXTA
 60 DATA 0 , 16 , 2 , 16 , 4 , 16 , 6 , 15 , 7 ,
    14 , 9 , 13 , 10 , 12 , 11 , 11
 70 DATA 12 , 10 , 13 , 8 , 14 , 6 , 15 , 4 , 15 ,
    2 , 15 , 0
100 FORX=38912+33*8TO38912+40*8 : READA : POKEX ,
    A : NEXTX
110 DATA 0 , 0 , 1 , 6 , 63 , 31 , 0 , 0
120 DATA 0 , 0 , 63 , 12 , 62 , 63 , 0 , 0
130 DATA 0 , 0 , 63 , 15 , 3 , 35 , 29 , 7
140 DATA 7 , 15 , 63 , 63 , 62 , 0 , 32
150 DATA 7 , 15 , 31 , 63 , 63 , 63 , 63 , 63
160 DATA 63 , 63 , 63 , 63 , 63 , 63 , 63 , 63
170 DATA 0 , 0 , 0 , 0 , 63 , 63 , 63 , 63
180 DATA 0 , 0 , 0 , 12 , 12 , 0 , 0 , 0
200 HIRES :  FORX=14TO36 : CURSETX , 199 , 1 :
    DRAWO , -199+186 , 1 : NEXTX
205 XT=186
210 FORX=14TOOSTEP-1 : CURSETX , 199 , 1
220 DRAWO , -199+XT , 1
230 XT=XT-(RND(1)*3-1)
240 NEXTX
245 XT=186
250 FORX=37TO61 : CURSETX , 199 , 1
260 DRAWO , -199+XT , 1
270 XT=XT-RND(1)*2
280 NEXTX
290 FORX=62TO90 : CURSETX , 199 , 1
300 DRAWO , -199+XT , 1
310 XT=XT-RND(1)*3+2
320 NEXTX
330 FORX=91TO160 : CURSETX , 199 , 1
340 XT=XT+RND(1)*3-1.5 : DRAWO , -199+XT , 1
350 NEXTX
360 FORX=161TO226 : CURSETX , 199 , 1
370 DRAWO , -199+XT , 1
380 NEXT
390 FORX=226TO239 : CURSETX , 199 , 1
400 XT=XT+RND(1)*3-3 : DRAWO , -199+XT , 1
410 NEXTX
420 PRINT : PRINT : PRINT
500 CURSET20 , 178 , 3 : CHAR38 , 0 , 1
510 CURMOV6 , 0 , 3 : CHAR38 , 0 , 1
520 CURMOV6 , 0 , 3 : CHAR38 , 0 , 1
530 CURMOV0 , -8 , 3 : CHAR39 , 0 , 1
```

```
 540 CURMOV-6 , 0 , 3 : CHAR39 , 0 , 1
 550 CURMOV-6 , 0 , 3 : CHAR38 , 0 , 1
 560 CURMOV0 , -8 , 3 : CHAR37 , 0 , 1
 580 GOTO1980
1000 REM
1010 IF XE=0 THEN RETURN
1090 IF P<0 THEN 1140
1100 IF ABS(XP+11-XE)>12 THEN 1140
1110 IF ABS(XP+4-YE)>9 THEN 1140
1120 TP=1 : NP=NP+1
1130 IF YE<150 THEN RETURN
1140 IF ABS(XE-26)>12 THEN RETURN
1150 EN=EN+1 : RETURN
1500 CURSETXP , YP , 3 : CHAR33 , 0 , C
1510 CURMOV6 , 0 , 3 : CHAR34 , 0 , C
1520 CURMOV6 , 0 , 3 : CHAR35 , 0 , C
1530 CURMOV6 , 0 , 3 : CHAR36 , 0 , C
1540 RETURN
1980 A=7 : TL=10 : TB=0 : TP=0 : NP=0 : TN=0 : TM=0 :
     P=-1 : B=0 : XN=0 : XM=0 : PP=0 : Y=186 : EN=-1
1990 VM=2 : VP=3 : KP=3 : KB=3 : KM=2 : GOTO3000
2000 I$=KEY$ : IFI$=""THEN2110
2010 I=ASC(I$)
2020 IFI=32THEN 3080 : REM" "
2030 IFI=11THENTL=TL+1 : GOTO2100 : REM'UP ARROW'
2040 IFI=10THENTL=TL-1 : GOTO2110 : REM'DOWN ARROW'
2050 IFI=8THEN3000 : REM'LEFT ARROW'
2060 IFI=9THEN3000 : REM'RIGHT ARROW'
2070 GOTO2110
2100 IFI=11THENTL=TL+1
2110 IFP>0THEN2170
2120 IFRND(1)<.49THEN2320
2130 PP=PP+1 : IFPP>20THEN4090
2140 P=1 : TP=0 : XP=206 : YP=RND(1)*40+30 : B=-1 :
     TB=0
2150 C=1 : GOSUB1500
2170 C=0 : GOSUB1500
2180 XP=XP-VP : YP=YP+KP*TP
2185 IF XP>220 THEN XP=220
2190 IF YP>191 THEN YP=191
2193 IFXP<0 THEN XP=0
2199 IFYP<0 THEN YP=0
2200 IFXP>1THEN2300
2210 C=0 : GOSUB1500
2220 P=-1 : IFB<0THENB=0
2230 GOTO2500
2300 C=1 : GOSUB1500
2310 IFPOINT(XP+26 , YP-2)=0ORPOINT(XP-2 ,
     YP+2)=0THEN2320
2315 XE=XP : YE=YP : GOSUB1000 :
     IFEN>0THEN4000 ELSE GOTO 2220
2320 IFB=0THEN2500
2330 IFB>0THEN2380
2340 IF RND(1)<.9 THEN 2500
2350 B=B+1 : XB=XP+10 : YB=YP+8 : TB=0
```

```
2360 CURSETXB , YB , 3 : CHAR42 , 0 , 1
2380 TB=TB+1
2390 CURSETXB , YB , 3 : CHAR42 , 0 , 0
2400 XB=XB-VP : YB=YB+KB*TB
2405 IF XB<0 THEN XB=0
2410 IFX<6THENB=0 : GOTO2500
2420 IFYB<190THEN2430
2425 B=0 : XE=XB : YE=190 : GOSUB1000 :
     IFEN>0THEN4000ELSEGOTO2500
2430 IFPOINT(XB , YB)=0THENCURSETXB , YB , 3 :
     CHAR42 , 0 , 1 : GOTO2500
2440 XE=XP : YE=YB : B=0
2450 GOSUB1000
2460 IFEN>0THEN4000
2500 IFXM=0THEN2700
2510 IFYM<10THEN2530
2520 CURSETXM , YM , 3 : CHAR46 , 0 , 0
2530 TM=TM+1 : XM=XM+VC : YM=YM-VS+KM*TM
2540 IFYM>195THENXM=0 : GOTO2700
2550 IFXM>233THENXM=0 : GOTO2700
2560 IFTM>=TLTHEN2620
2570 IFYM<10THEN2700
2580 IFPOINT(XM , XY)<>0THEN2620
2590 IFP<0THEN2660
2600 IFABS(XP+11-XM)>12THEN2660
2610 IFABS(YP+4-YM)>9THEN2660
2620 XE=XM : YE=YM : XM=0
2630 IFYM<10THEN2700
2640 GOSUB1000 : IFEN>0THEN4000
2650 GOTO2700
2660 IFYM<10THEN2700
2670 CURSETXM , YM , 3 : CHAR46 , 0 , 1
2700 IFXN=0THEN2880
2710 IFYN<10THEN2730
2720 CURSETXN , YN , 3 : CHAR46 , 0 , 0
2730 TN=TN+1 : XN=XN+UC : YN=YN-US+KM*TN
2740 IFYN>195THENXN=0 : GOTO2880
2750 IFXN>233THENXN=0 : GOTO2880
2760 IFTN>=TLTHEN2820
2770 IFYN<10THEN2880
2780 IFPOINT(XN , YN)<>0THEN2820
2790 IFP>0THEN2860
2800 IFABS(XP+11-XN)>12THEN2860
2810 IFABS(YP+4-YN)>9THEN2860
2820 XE=XN : YE=YN : XN=0
2830 IFYN<10THEN2880
2840 GOSUB1000 : IFEN>0THEN4000
2850 GOTO2880
2860 IFYN<10THEN2880
2870 CURSETXN , YN , 3 : CHAR46 , 0 , 1
2880 IFP<0THEN2000
2890 IFPOINT(XP+3 , YP+8)<>0THEN2910
2900 IFPOINT(XP+15 , YP+8)=0THEN2000
2910 XE=XP+11 : YE=YP+6 : P=-1 : GOSUB1000 :
     IFEN>0THEN4000
```

```
2920 GOTO2000
3000 CURSET26 , 172 , 3 : DRAWC(A) , -S(A) , 0
3010 IFI=9THENA=A-1 : IFA<0THENA=0
3020 IFI=8THENA=A+1 : IFA>13THENA=13
3030 CURSET26 , 172 , 3 : DRAWC(A) , -S(A) , 1
3040 GOTO2110
3080 IFXM<>0THEN3100
3085 SHOOT
3090 CM=C(A) : SM=S(A) : XM=30+CM : YM=166-SM :
     TM=0 : VC=CM*VM : VS=SM*VM
3095 CURSETXM , YM , 3 : CHAR46 , 0 , 1 : GOTO2110
3100 IFXN<>0THEN2110
3105 SHOOT
3110 CN=C(A) : SN=S(A) : XN=30+CN : YN=166-SN :
     TN=0 : UC=CN*VM : US=SN*VM
3115 CURSETXN , YN , 3 : CHAR46 , 0 , 1 : GOTO2110
4000 REM        ;
4005 EXPLODE
4007 I$=KEY$ : WAIT100
4010 TEXT : CLS : PLOT 5 , 0 , "KILLED IN ACTION"
4020 PLOT 5 , 2 , "SCORE :  "+RIGHT$(STR$(NP) ,
     LEN(STR$(NP))-1)+" HITS"
4030 IF NP<10 THEN 4120
4040 IF NP>15 THEN 4070
4050 PLOT 5 , 4 , "POSTED A MEDAL"
4060 GOTO 4120
4070 PLOT 5 , 4 , "NATIONAL HERO"
4080 GOTO 4120
4090 CLS
4100 PLOT 5 , 0 , "RETURNING SOLDIER"
4110 GOTO 4020
4120 I$=KEY$ : IF I$<>"" THEN 200 ELSE 4120
4130 IF I$<>"" THEN 200
4140 GOTO 4120
4150 GOTO 4150
5000 END
```

Freeway Frog



Poor Freddy Frog and his family want to get home but the busy freeway is in between them and their home. To get home they must dodge trucks, cars and motorbikes. Motorbikes are the fastest but occasionally a very high speed police car will wail its way across the screen. You use the cursor control keys to move the frogs in the desired direction. 10 points are scored for moving up and down and 100 points are scored for getting a frog home (the top of the screen).

This program's value is in showing how to make many objects move at once and how to move them at different speeds. However it also demonstrates the lack of ability to control graphics in every detail and still use more than 2 colours. This does not need to be a problem though - you just have to design games that don't have large objects colliding or many different coloured objects moving all over the screen. This game also demonstrates the limitations of BASIC with regard to speed.

HOW THE PROGRAM WORKS

This program has one main loop but not everything is updated in each pass. In fact moving the motor-bikes and testing for collisions are the only ones. Other routines are only called when certain variables tell them to do so. The moving and drawing of frogs and cars is done every second loop and the trucks are done every third loop. This is to obtain a difference in relative "speed" between the different vehicles. If you wanted to change the speeds, increase the value of the variables in lines 310 (for cars) and 910 (for trucks).

Whenever a frog is moved a check is made to see if it was on top of any part of the road. If it was on the road then that part is redrawn as the frog erases anything under it. This is because of the attributes on the area occupied by the frog having been set.

By rewriting the main loop in Machine Code the game could be speeded up considerably. If you have mastered machine code this would be an ideal game to try it on as the end result will astound you.

STRUCTURE	LINE(S)
INITIALIZE - SCREEN	5
- GRAPHICS CHARACTERS	10-30
- VARIABLES	40-50
- SCREEN	70-90
- POSITION OF VEHICLES	100-150
DRAW HIGHWAY	160-180
DRAW INITIAL FROGS	200-210
DRAW SCORE LINE	220
HAVE POLICE CAR GO THROUGH OCCASIONALLY	230-300
CHECK FOR COLLISION	500-550
SCAN KEYBOARD - IF BUTTON PUSHED - MOVE	600-660
CALL MOVE ROUTINES - BIKES EVERY CYCLE - CARS EVERY 2ND CYCLE - TRUCKS EVERY 3RD CYCLE	700-950
REMOVE FROG FROM SCREEN AND REPLACE PART OF ROAD IF NECESSARY	990-997
FOUR ROUTINES TO DRAW THE FOUR FROGS	1000-1079
TWO ROUTINES TO DRAW TWO CARS	1080-1170
TWO ROUTINES TO DRAW TWO BIKES	1200-1295
TWO ROUTINES TO DRAW TWO TRUCKS	1300-1395
MOVE AND CALL DRAW ROUTINES FOR CARS	2000-2012
MOVE AND CALL DRAW ROUTINES FOR BIKES	2100-2112
MOVE AND CALL DRAW ROUTINES FOR TRUCKS	2200-2212
ROUTINE IF A FROG IS "SPLATTERED"	4000-4010
ROUTINE TO FINISH GAME	4015-4490
ROUTINE IF FROG GETS HOME	5000-5030
DATA TO DEFINE GRAPHIC CHARACTERS	9000-9235

VARIABLES

HI = HIGH SCORE
SC = SCORE
M1, M2 = POSITION OF BIKES
C1, C2 = POSITION OF CARS
T1, T2 = POSITION OF TRUCKS
PX, PY = POSITION OF FROG
PZ = CONTROL FROG MOVEMENT
PA = DIRECTION FROG IS POINTING
X = GENERAL POSITION
FC = FROGS LEFT
F = FROG COLOUR
C = CAR COLOUR
CE, CF = CAR COLOUR

```
100 REM * FREEWAY FROG *
110 REM
120 PAPER0 : INK2 : PRINTCHR$(20)CHR$(6)CHR$(17)
130 FORI=47624TO47951
140 READA : POKEI , A
150 NEXT
160 HI=0
170 FORI=1TO37 : L1$=L1$+CHR$(69) :
    IFI<18THENL2$=L2$+CHR$(69)+" "
180 NEXTI
190 SC=0 : CLS
200 FORI=3TO25
210 PLOT0 , I , CHR$(9)
220 NEXTI
230 M1=2*INT(RND(1)*15) : M2=2*INT(RND(1)*15)
240 C1=INT(RND(1)*30) : C2=INT(RND(1)*30)
250 T1=INT(RND(1)*30) : T2=INT(RND(1)*30)
260 CE=2 : CF=6
270 PZ=2 : TZ=2 : CZ=1
280 F=4 : FC=5
290 GOSUB300 : GOSUB310 : GOSUB330 : GOTO340
300 PLOT1 , 6 , 3 : PLOT2 , 6 , L1$ : RETURN
310 PLOT2 , 12 , L2$ : RETURN
320 FORI=3TO37STEP2 : PLOTI , 12 , " " : NEXTI :
    RETURN
330 PLOT1 , 18 , 3 : PLOT2 , 18 , L1$ : RETURN
340 PY=22 : PX=20 : F=2
350 FORX=3TOFC*3+1 STEP3 : PX=X : GOSUB900 : NEXTX
360 S$=" Score  : "+STR$(SC)+"        High Score  :
    "+STR$(HI) : PLOT2 , 0 , S$
370 IF RND(1)<.99THEN GOTO490
380 SOUND2 , 30000 , 1
390 IF RND(1)>.5 THEN GOTO440
400 C=5 : MUSIC1 , 5 , 1 , 0 : PLAY1 , 0 , 4 , 900
410 FORX=-1TO37 : GOSUB1110 :
    IFPY=14ANDPX=XTHENGOTO1650
420 NEXTX
430 PLAY0 , 0 , 0 , 0 : GOTO490
440 C=5 : MUSIC1 , 5 , 1 , 0 : PLAY1 , 0 , 4 , 900
450 FORX=37TO-1STEP-1 : GOSUB1190 :
    IFPY=9ANDPX=XTHENGOTO1650
460 NEXTX
470 PLAY0 , 0 , 0 , 0
480 GOTO560
490 IFPY>17ORPY<5THENGOTO560
500 IF(PY>=14ANDPY<=15ANDPX+1>=C1ANDPX<=C1+4)
    THENGOTO1650
510 IFPY>=15ANDPX+1>=T1ANDPX<=T1+6THENGOTO1650
520 IF(PY>=12ANDPY<=14ANDPX+1>=M1ANDPX<=M1+1)
    THENGOTO1650
530 IF(PY>=9ANDPY<=11ANDPX+1>=T2ANDPX<=T2+6)
    THENGOTO1650
540 IF(PY>=8ANDPY<=9ANDPX+1>=C2ANDPX<=C2+4)
    THENGOTO1650
```

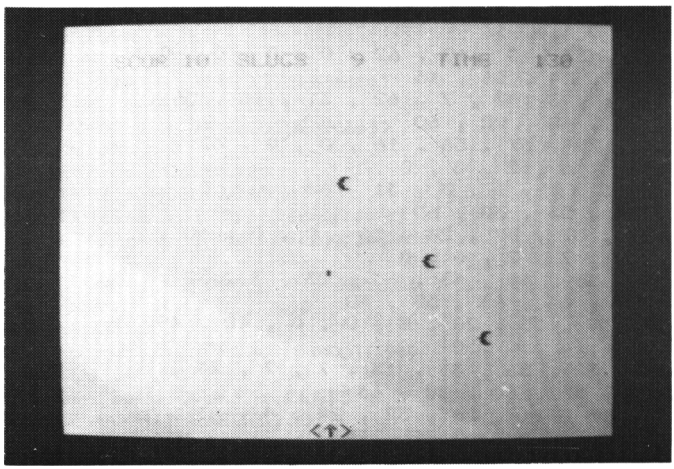
```
550 IF(PY<=8ANDPX+1>=M2ANDPX<=M2+1)THENGOTO1650
560 IF PZ>0 THEN PZ=PZ-1 : GOTO690
570 PZ=1 : Z=PEEK(520) : IFZ=56THENGOTO700
580 SOUND1 , 10000 , 0 : MUSIC1 , 1 , 5 , 0 : PLAY1 ,
    0 , 1 , 600
590 GOSUB760
600 PY=PY-(Z=180)+(Z=156)
610 PX=PX+(Z=172) - (Z=188)
620 IFZ=156THENPA=0
630 IFZ=188THENPA=20
640 IFZ=180THENPA=40
650 IFZ=172THENPA=60
660 GOSUB830+PA
670 IF PY=3 THEN GOSUB1790
680 SC=SC-10*(Z=180 OR Z=156)
690 REM
700 GOSUB1550
710 IF CZ THEN CZ=CZ-1 : GOTO730
720 GOSUB1500 : CZ=1
730 IF TZ>0 THEN TZ=TZ-1 : GOTO750
740 GOSUB1600 : TZ=2
750 GOTO360
760 P1=PY
770 IFP1=6THENGOSUB300 : GOTO810
780 IFP1=12THENGOSUB310 : GOTO810
790 IFP1=18THENGOSUB330 : GOTO810
800 PLOTPX , P1 , " " : PLOTPX+1 , P1 , " "
810 IFP1=PYTHENP1=P1+1 : GOTO770
820 RETURN
830 PLOTPX-1 , PY , CHR$(F)+"Z["
840 PLOTPX-1 , PY+1 , CHR$(F)+"\]"
850 IF PY=6 OR PY=18 THEN PLOTPX+2 , PY , CHR$(3) :
    RETURN
860 IF PY+1=6 OR PY+1=18 THEN PLOTPX+2 , PY+1 ,
    CHR$(3) : RETURN
870 IF PY=12 THEN PLOTPX+2 , PY , 7 : RETURN
880 IF PY+1=12 THEN PLOTPX+2 , PY+1 , 7 : RETURN
890 RETURN
900 PLOTPX-1 , PY , F : PLOTPX , PY , "^" :
    PLOTPX+1 , PY , "£"
910 PLOTPX-1 , PY+1 , F : PLOTPX , PY+1 ,
    CHR$(96) : PLOTPX+1 , PY+1 , "a"
920 IF PY=6 OR PY=18 THEN PLOTPX+2 , PY , CHR$(3) :
    RETURN
930 IF PY+1=6 OR PY+1=18 THEN PLOTPX+2 , PY+1 ,
    CHR$(3) : RETURN
940 IF PY=12 THEN PLOTPX+2 , PY , 7 : RETURN
950 IF PY+1=1 THEN PLOTPX+2 , PY+1 , 7 : RETURN
960 RETURN
970 PLOTPX-1 , PY , F : PLOTPX , PY , "b" :
    PLOTPX+1 , PY , "c"
980 PLOTPX-1 , PY+1 , F : PLOTPX , PY+1 , "d" :
    PLOTPX+1 , P+1 , "e"
990 IF PY=6 OR PY=18 THEN PLOTPX+2 , PY , CHR$(3) :
    RETURN
```

```
1000 IF PY+1=6 OR PY+1=18 THEN PLOTPX+2 , PY+1 ,
     CHR$(3) : RETURN
1010 IF PY=12 THEN PLOTPX+2 , PY , 7 : RETURN
1020 IF PY+1=12 THEN PLOTPX+2 , PY+1 , 7 : RETURN
1030 RETURN
1040 PLOTPX-1 , PY , F : PLOTPX , PY , "f" :
     PLOTPX+1 , PY , "g"
1050 PLOTPX-1 , PY+1 , F : PLOTPX , PY+1 , "h" :
     PLOTPX+1 , PY+1 , "i"
1060 IF PY=6 OR PY=18 THEN PLOTPX+2 , PY , CHR$(3) :
     RETURN
1070 IF PY+1=6 OR PY+1=18 THEN PLOTPX+2 , PY+1 ,
     CHR$(3) : RETURN
1080 IF PY=12 THEN PLOTPX+2 , PY , 7 : RETURN
1090 IF PY+1=12 THEN PLOTPX+2 , PY+1 , 7 : RETURN
1100 RETURN
1110 S$=" JKLM" : S2=15
1120 IF X>3 THENGOTO1150                    1550
1130 PLOT2 , S2 , CHR$(C) : PLOTX+5 , S2 , CHR$(F)
1140 S0=3 : S$=RIGHT$(S$ , X+2) : GOTO1180
1150 IFX>33THENS0=X : S$=LEFT$(S$ , 38-X) : PLOTX-1 ,
     S2 , CHR$(C) : GOTO1180
1160 S0=X : S1=6
1170 PLOTX-1 , S2 , CHR$(C) : PLOTX+5 , S2 , CHR$(F)
1180 PLOTS0 , S2 , S$ : RETURN
1190 S$="NOPQ " : S2=9
1200 GOTO1120
1210 IFX>3THENGOTO1240
1220 PLOT2 , S2 , CHR$(7) : PLOTX+3 , S2 , CHR$(F)
1230 S0=3 : S$=RIGHT$(S$ , X) : GOTO1270
1240 IFX>35THENS0=X : S$=LEFT$(S$ , 38-X) : PLOTX-1 ,
     S2 , CHR$(7) : GOTO1270
1250 S0=X : S1=4
1260 PLOTX-1 , S2 , CHR$(7) : PLOTX+1 , S2 , CHR$(F)
1270 PLOTS0 , S2 , S$ : RETURN
1280 S$=" RS" : S2=13 : GOSUB1210
1290 S$=" TU" : S2=14
1300 GOTO1210
1310 S$="VW " : S2=7
1320 GOSUB1210
1330 S$="XY " : S2=8
1340 GOTO1210
1350 IFX>3THENGOTO1380
1360 PLOT2 , S2 , CHR$(4) : PLOTX+7 , S2 , CHR$(F)
1370 S0=3 : S$=RIGHT$(S$ , X+5) : GOTO1410
1380 IFX>31THENS0=X : S$=LEFT$(S$ , 38-X) : PLOTX-1 ,
     S2 , CHR$(4) : GOTO1410
1390 S0=X : S1=8
1400 PLOTX-1 , S2 , CHR$(4) : PLOTX+6 , S2 , CHR$(F)
1410 PLOTS0 , S2 , S$ : RETURN
1420 S$=" AAAAAB" : S2=16
1430 GOSUB1350
1440 S$=" CDEEDF" : S2=17
1450 GOTO1350
1460 S$="GAAAAA " : S2=10
1470 GOSUB 1350
```

```
1480 S$="HDEEDI " : S2=11
1490 GOTO1350
1500 C=CE : X=C1 : GOSUB1110
1510 C=CF : X=C2 : GOSUB1190
1520 C1=C1+1 : IFC1>37THENC1=-2
1530 C2=C2-1 : IFC2<-1THENC2=37
1540 RETURN
1550 X=M1 : GOSUB1280
1560 M1=M1+1 : IFM1>37THENM1=1
1570 X=M2 : GOSUB1310
1580 M2=M2-1 : IFM2<0THENM2=37
1590 RETURN
1600 X=T1 : GOSUB1420
1610 X=T2 : GOSUB1460
1620 T1=T1+1 : IFT1>37THENT1=-3
1630 T2=T2-1 : IFT2<-4THENT2=37
1640 RETURN
1650 SOUND1 , 10000 , 0 : MUSIC1 , 3 , 5 , 0 :
     PLAY7 , 0 , 2 , 500
1660 GOSUB760
1670 FC=FC-1 : IFFC>0THENGOTO340
1680 CLS
1690 S$="Game Over" : PLOT3 , 3 , S$
1700 IF SC<HITHENGOTO130
1710 S$="But your score is the new"+CHR$(12)+"HIGH
     SCORE" : PLOT2 , 5 , S$
1720 HI=SC
1730 S$="HIT 'Y' to play again" : PLOT5 , 7 , S$
1740 GET A$
1750 IF A$="y" THEN GOTO190
1760 IF A$<>"n" THEN GOTO1740
1770 PRINTCHR$(20)CHR$(6)CHR$(17)
1780 END
1790 SOUND1 , 500 , 0 : PLAY1 , 0 , 1 , 1 :
     MUSIC1 , 1 , 1 , 10 : WAIT100 :
     MUSIC1 , 2 , 1 , 10 : WAIT200
1800 MUSIC1 , 3 , 1 , 10 : WAIT100 :
     MUSIC1 , 2 , 4 , 10 : WAIT100
1810 PLAY0 , 0 , 0 , 0
1820 GOSUB760
1830 SC=SC+100
1840 FC=FC-1 : IFFC>0THEN GOTO 340
1850 GOTO 1680
1860 REM TRUCK - DATA LINES 9000-9015
1870 REM CAR   - DATA LINES 9050-9065
1890 REM BIKIES- DATA LINES 9100-9115
1900 REM FROGS - DATA LINES 9200-9235
1910 DATA63 , 63 , 63 , 63 , 63 , 63 , 63 , 63 , 0 ,
     32 , 30 , 25 , 25 , 25 , 31 , 31
1920 DATA63 , 63 , 63 , 63 , 18 , 23 , 2 , 0 , 63 ,
     63 , 63 , 63 , 8 , 28 , 8 , 0 , 63 , 63 , 63 ,
     63 , 0 , 0 , 0 , 0
1930 DATA31 , 31 , 30 , 63 , 21 , 46 , 4 , 0 , 0 ,
     1 , 30 , 38 , 38 , 38 , 62 , 62
```

```
1940 DATA62 , 62 , 30 , 63 , 42 , 29 , 8 , 0 , 63 ,
     63 , 63 , 63 , 9 , 29 , 8 , 0
1950 DATA0 , 1 , 3 , 31 , 62 , 61 , 11 , 1 , 60 ,
     34 , 33 , 48 , 31 , 47 , 32 , 0
1960 DATA0 , 0 , 0 , 63 , 63 , 58 , 23 , 2 , 0 , 0 ,
     0 , 48 , 56 , 16 , 0 , 0
1970 DATA0 , 0 , 0 , 3 , 7 , 2 , 0 , 0 , 0 , 0 , 0 ,
     63 , 47 , 23 , 41 , 16
1980 DATA30 , 17 , 33 , 3 , 63 , 62 , 0 , 0 , 0 , 0 ,
     48 , 62 , 31 , 54 , 56 , 16
1990 DATA0 , 0 , 1 , 3 , 7 , 7 , 62 , 31 , 16 , 56 ,
     48 , 32 , 32 , 48 , 28 , 60
2000 DATA15 , 7 , 29 , 57 , 56 , 16 , 0 , 0 , 62 ,
     54 , 39 , 39 , 6 , 2 , 0 , 0
2010 DATA2 , 7 , 3 , 1 , 1 , 3 , 31 , 63 , 0 , 0 ,
     32 , 48 , 56 , 56 , 30 , 60
2020 DATA31 , 27 , 57 , 57 , 24 , 16 , 0 , 0 , 60 ,
     56 , 46 , 39 , 7 , 2 , 0 , 0
2030 DATA1 , 19 , 53 , 39 , 63 , 63 , 23 , 3 , 32 ,
     50 , 43 , 57 , 63 , 63 , 58 , 48
2040 DATA3 , 3 , 39 , 55 , 20 , 8 , 0 , 0 , 48 , 48 ,
     57 , 59 , 10 , 4 , 0 , 0
2050 DATA4 , 24 , 32 , 56 , 31 , 15 , 7 , 7 , 24 ,
     52 , 2 , 48 , 56 , 60 , 58 , 63
2060 DATA7 , 7 , 15 , 31 , 56 , 32 , 24 , 4 , 63 ,
     58 , 60 , 56 , 48 , 22 , 52 , 24
2070 DATA0 , 0 ,  , 28 , 23 , 55 , 35 , 3 , 0 , 0 ,
     4 , 14 , 58 , 59 , 9 , 48
2080 DATA3 , 23 , 63 , 63 , 39 , 53 , 19 , 1 , 48 ,
     58 , 63 , 63 , 57 , 43 , 50 , 32
2090 DATA0 , 4 , 30 , 19 , 2 , 15 , 23 , 63 , 16 ,
     28 , 6 , 2 , 14 , 60 , 56 , 48
2100 DATA63 , 23 , 15 , 2 , 19 , 30 , 4 , 0 , 48 ,
     56 , 60 , 14 , 2 , 6 , 28 , 16
2110 PRINTX , I , S0 , S1 , S2 , S$
2120 IF PY=7 OR PY=17 THEN PLOTPX+2 , PY , 3 : RETURN
2130 PLOTX , Y , CHR$(C) : PLOTX+5 , Y , CHR$(F)
```

Duck Shoot



The game works on a similar principle to the duck shoots at carnivals.

You begin with your rifle positioned at the bottom centre of the screen. You may change the horizontal position of your rifle at any time during the game.

The object of the game is to hit as many ducks as possible using only nine "slugs". Once all nine "slugs" have been used, the game is reset and you try again. Each duck shot is worth ten points. Therefore, the maximum amount of points is ninety. It takes a lot of practice to reach this score.

Good luck!

STRUCTURE	LINE(S)
INITIALIZATION	7-70
MAIN LOOP	80-140
GAME RESET	150-180
LAUNCH BIRD	1000-1080
CHECK KEY PRESSED AND ACT	1100-1180
FIRE SLUG	1200-1260
MOVE SLUG	1300-1390
MOVE BIRDS	1500-1630
CHECK IF DUCK HIT	1720
UPDATE SCORE	1760

VARIABLES

```
SC = Score
T = Time
A = Slugs fired
FIN = End-of-game flag
PO = Position of rifle
ND = Number of ducks on screen
CD = Current duck being moved
RO = Random UP/DOWN/STRAIGHT duck direction flag
R1 = Random vertical duck starting position
B(,) = Array for holding each bird's (horizontal,
       vertical) position
K = Keyboard value PEEKed from (520)
MX = Last horizontal position of Slug (-1 means no
     moving slug)
MY = Last vertical position of Slug
I = Pointer to bird array
X = Current horizontal position of slug/bird
Y = Current vertical position of slug/bird
```

SPECIAL NOTES

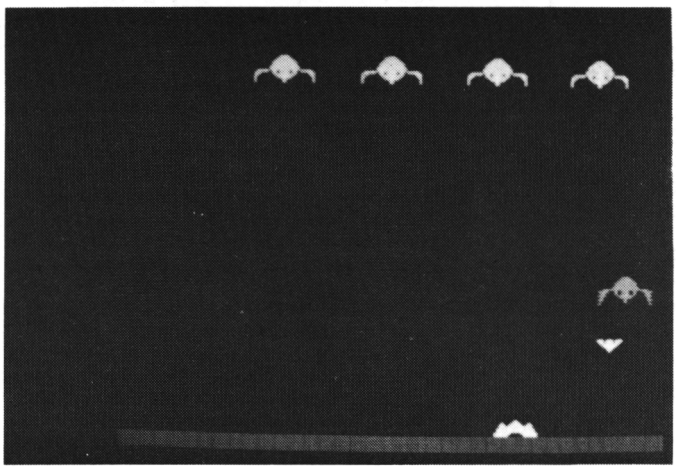
The 'duck shoot' program is a good example of modular programming. Each task within the program is a separate subroutine and each subroutine is called separately from the main loop.

If you go to the trouble of highlighting the start of each routine as set out in the structure, the program will show itself as a set of subroutines set out sequentially throughout the program.

```
  5 REM * DUCK SHOOT *
  6 REM
  7 POKE 618 , 10
 10 CLS
 20 SC=0 : T=0 : A=0 : FIN=0 : MX=-1
 30 PLOT 2 , 2 , "SCORE" : PLOT  7 , 2 , STR$(SC)
 40 PLOT 12 , 2 ,  "SLUGS " : PLOT 19 , 2 , STR$(A)
 50 PLOT 26 , 2 , "TIME" : PLOT 32 , 2 , STR$(T)
 60 PO=18 : PLOT17 , 26 , "<^>"
 70 DIMB(2 , 1) : ND=0 : CD=0
 80 REPEAT
 90 IF ND<3THENGOSUB1000 : REM LAUNCH BIRD
100 GOSUB1100 : REM CHECK KEY AND ACT
110 IF MX<>-1 THEN GOSUB 1300 : REM MOVE BULLET
120 T=T+1 : PLOT32 , 2 , STR$(T)
130 IFND<>0THENGOSUB1500 : REM MOVE BIRDS
140 UNTILFIN=1
150 WAIT200 : CLS : PLOT3 , 10 , "Another game ?" :
    GET K$
160 IF K$="Y"THEN RUN
170 IF K$<>"N"THENGOTO150
180 END
1000 REM LAUNCH BIRD
1010 R0=RND(1)*4
1020 IFR0>1THEN RETURN
1030 ND=ND+1
1040 R1=RND(1)*20
1050 B(CD , 0)=38 : B(CD , 1)=INT(R1)+4
1060 PLOT38 , B(CD , 1) , "{" : CD=CD+1
1080 RETURN
1100 REM CHECK KEY & ACT
1110 K=PEEK(520)
1111 IF K=174 THEN 2000
1120 IFK=132THENGOSUB1200 : RETURN
1130 IFK=172THENIFPO>4THENPO=PO-1 : GOTO1170
1140 IFK<>188THEN RETURN
1150 IFPO<38 THENPO=PO+1
1170 PLOTPO-2 , 26 , " <> "
1180 RETURN
1200 REM LAUNCH BULLET
1210 IF A=9 THEN RETURN
1215 IF MX<>-1 THEN PLOT MX , MY , " "
1220 SHOOT
1230 A=A+1
1240 MX=PO : MY=25
1250 PLOT 19 , 2 , STR$(A)
1260 RETURN
1300 REM MOVE BULLET
1320 IFMX=-1THEN RETURN
1325 X=MX : Y=MY
1330 PLOTX , Y , " "
1340 IF Y>3 THEN 1360
1345 SHOOT : WAIT 5 : PLAY 0 , 0 , 0 , 0
```

```
1350 IFA=9THENFIN=1
1355 MX=-1 : RETURN
1360 Y=Y-1 : MY=Y
1365 GOSUB1700
1370 PLOTX , Y , "?"
1390 RETURN
1500 REM MOVE BIRDS
1510 FORI=0TOND-1
1520 RO=INT(RND(1)*4)+1
1530 X=B(I , 0) : Y=B(I , 1)
1540 PLOTX , Y , " "
1550 IF X=2THEN CD=I : GOTO 1040
1560 X=X-1 : B(I , 0)=X
1565 IF(Y=3 OR Y=23) GOTO1610
1570 ON RO GOTO1580 , 1610 , 1610 , 1600
1580 Y=Y+1 : B(I , 1)=Y
1590 GOTO1610
1600 Y=Y-1 : B(I , 1)=Y
1610 PLOT X , Y , "(" : IF X=MX AND Y=MY THEN 1740
1620 NEXTI
1630 RETURN
1700 REM CHECK IF DUCK HIT
1705 X=MX : Y=MY
1720 IF SCRN(X , Y)<>123 THEN RETURN
1740 PLOT X , Y , "*"
1750 PING
1755 WAIT 30
1760 SC=SC+10 : PLOT7 , 2 , STR$(SC)
1770 IF A=9 THEN FIN=1 :RETURN
1780 MX=-1
1795 RETURN
```

Alien Blitz



You are the only defender of the space station. Suddenly, overhead, they appear - a convoy of hostile aliens, hovering just out of your laser's reach.

One alien swoops - flapping his wings to gain speed, firing missiles as he comes - boy, he's quick; got to get that laser lined up - got him! But wait... here comes another one...

Your controls are: cursor control keys for left and right; space bar to fire.

HOW THE PROGRAM WORKS

This is an action game and therefore the main programming considerations are speed and visual effects. Sound is only used where necessary, as it slows the running of a program.

To make the program even faster, two main loops are used with a minimum of variables and IF statements. To achieve the flapping wings effect one loop has wings up and the other has wings down.

STRUCTURE	LINE(S)
INITIALIZE SCREEN, DEFINE SHAPES, DRAW CONVOY	130-230
INITIALIZE VARIABLES	240-280
FIRST CYCLE	300-440
SECOND CYCLE	540-770
YOU HIT ALIEN	790-890
ALIEN MISSILE REACHES YOU	910-980
ALIEN REACHES YOU	1010-1110
DESTRUCTION OF ENEMY CONVOY	1120-1180

VARIABLES

X, Y = POSITION OF ATTACKING ALIEN
G = YOUR POSITION
S = NEXT ALIEN TO ATTACK

```
100 REM * ALIEN BLITZ *
110 REM
120 PRINTCHR$(17)CHR$(6)CHR$(20)
130 CLS : PAPER0
140 RESTORE : GOSUB1220
150 FORI=0TO25 : PLOT0 , I , CHR$(9) : NEXTI
160 PLOT1 , 1 , CHR$(3) : PLOT1 , 2 , CHR$(3)
170 FORI=3TO22 : PLOT1 , I , CHR$(1) : NEXTI
180 PLOT1 , 23 , CHR$(6) : PLOT1 , 24 , CHR$(4)
190 FORZ=4TO32STEP7
200 :  PLOTZ+1 , 1 , "AB"
210 :  PLOTZ , 2 , "ECDF"
220 NEXTZ
230 FORI=2TO38 : PLOTI , 24 , "R" : NEXTI
240 REM
250 S=0
260 G=5
270 X=4+7*S
280 Y=2
290 PLOTX , 0 , "      " : PLOTX , 1 , "      "
300 G=G+(G<34)*(PEEK(520)=188)-(G>2)*(PEEK(520)=172)
310 PLOTG , 23 , " OPQ "
320 PLOTX , Y , "      " : PLOTX , Y+1 , "      "
330 X=X+(Y>3)-(Y<4)+2*SGN(G-X)*(RND(1)>.5)
340 IFX>33THENX=2
350 IFX<2THENX=33
360 Y=Y+1
370 PLOTX , Y , " GABH " : PLOTX , Y+1 , "  CD "
380 IFY=21THENGOTO1010
390 IF RND(1)>.15OR Y>19THENGOTO470
400 SHOOT
410 FORI=Y+2TO21
420 :  PLOT1 , I , CHR$(1) : PLOT1 , I+1 , CHR$(7)
430 :  PLOTX+1 , I , "  " : PLOTX+1 , I+1 , "MN"
440 NEXTI
450 PLOT1 , 22 , CHR$(1) : PLOTX+1 , 22 , "  "
460 IF X>=G AND X<G+3 THEN GOTO910
470 IF PEEK(520)<>132 THEN GOTO540
480 ZAP
490 FORI=22TO-1STEP-1
500 IF I>3 THENPLOT1 , I , CHR$(3) : PLOTG+2 , I ,
    "S"
510 IF I<19 THEN PLOT1 , I+4 , CHR$(1) : PLOTG+2 ,
    I+4 , " "
520 NEXTI
530 IF (X>G-3 AND X<G+2)THENGOTO790
540 G=G+(G<34)*(PEEK(520)=188)-(G>2)*(PEEK(520)=172)
550 PLOTG , 23 , " OPQ "
560 PLOTX , Y , "      " : PLOTX , Y+1 , "      "
570 X=X+(Y>3)-(Y<4)+2*SGN(G-X)*(RND(1)>.5)
580 IFX>33THENX=2
590 IFX<2THENX=33
600 Y=Y+1
```

(5#) — line 280 to line 290

(6#) — line 310 to line 320

(5#) — line 320 to line 330

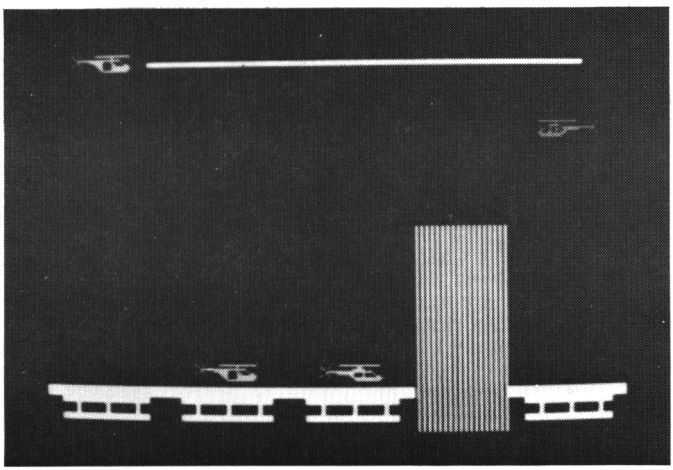
(5#) — line 550 to line 560

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

```
610 PLOTX , Y , "  AB  " : PLOTX , Y+1 , " ICDJ"
620 IFY=21THENGOTO1010
630 IF RND(1)>.150R Y>19THENGOTO710
640 SHOOT
650 FORI=Y+2TO21
660  :   PLOT1 , I , CHR$(1) : PLOT1 , I+1 , CHR$(7)
670  :   PLOTX+1 , I , "  " : PLOTX+1 , I+1 , "MN"
680 NEXTI
690 PLOT , 22 , CHR$(1) : PLOTX+1 , 22 , "  "
700 IF X>=G AND X<G+3 THEN GOTO910
710 IF PEEK(520)<>132 THEN GOTO300
720 ZAP
730 FORI=22TO-1STEP-1
740 IF I>3 THENPLOT1 , I , CHR$(3) : PLOTG+2 , I ,
    "S"
750 IF I<19 THEN PLOT1 , I+4 , CHR$(1) : PLOTG+2 ,
    I+4 , " "
760 NEXTI
770 IF (X>G-3 AND X<G+2)THENGOTO790
780 GOTO300
790 FORY=Y+1TO21
800 WAIT10
810 :  PLOTX , Y-1 , "     "
820 :  PLOTX , Y , "  AB  "
830 :  PLOTX , Y+1 , " KCDL "
840 NEXTY
850 PLOTX , Y-1 , "     "
860 S=S+1
870 PLOTX , Y , "     "
880 PLOTX , Y+1 , "     "
890 IFS=5THENGOTO1120
900 GOTO270
910 EXPLODE
920 FORI=1TO100
930 :  IFZ=0THENZ=7ELSEZ=0
940 :  PAPERZ
950 NEXTI
960 PAPER0
970 WAIT 50
980 PLOTG , 23 , "     "
990 PLOTX , Y , "     " : PLOTX , Y+1 , "     "
1000 GOTO250
1010 PLOTX , Y , "     " : PLOTX , Y+1 , "     "
1020 PLOTG , 23 , "     "
1030 IF (X>G-3 AND X<G+2)THENGOTO1050
1040 GOTO270
1050 EXPLODE
1060 FORI=7TO0STEP-1
1070 :  PAPERI
1080 NEXTI
1090 PAPER0
1100 WAIT50
1110 CLS : GOTO150
1120 CLS
1130 PLOT7 , 10 , "Congratulations - you have"
1140 PLOT3 , 11 , "destroyed the entire enemy fleet"
```

```
1150 WAIT300
1160 REM MUSIC
1170 PLOT1 , 15 , CHR$(12)+CHR$(5)+"WATCH OUT - HERE
     THEY COME AGAIN"
1180 WAIT200
1190 CLS : GOTO150
1200 PRINT : PRINT : PRINT : PRINT
1210 END
1220 FORI=47624TO47624+(19*8)-1
1230 :  READX : POKEI , X
1240 NEXTI
1250 DATA0 , 0 , 3 , 7 , 15 , 31 , 31 , 63 , 0 , 0 ,
     48 , 56 , 60 , 62 , 62 , 63
1260 DATA63 , 27 , 27 , 15 , 7 , 1 , 1 , 0 , 63 ,
     54 , 54 , 60 , 56 , 32 , 32 , 0
1270 DATA0 , 31 , 56 , 48 , 32 , 32 , 32 , 0 , 0 ,
     62 , 7 , 3 , 1 , 1 , 1 , 0
1280 DATA48 , 24 , 56 , 28 , 60 , 28 , 62 , 15 , 3 ,
     6 , 7 , 14 , 15 , 14 , 31 , 60
1290 DATA15 , 62 , 28 , 60 , 24 , 56 , 16 , 48 , 60 ,
     31 , 14 , 15 , 6 , 7 , 2 , 3
1300 DATA6 , 5 , 4 , 44 , 40 , 16 , 0 , 0 , 24 , 40 ,
     8 , 13 , 5 , 2 , 0 , 0
1310 DATA0 , 59 , 27 , 15 , 7 , 3 , 1 , 0 , 0 , 46 ,
     44 , 56 , 48 , 32 , 0 , 0
1320 DATA0 , 4 , 14 , 15 , 31 , 31 , 63 , 63 , 12 ,
     30 , 63 , 63 , 63 , 63 , 33 , 0 , 0 , 8 , 28 ,
     60 , 62 , 62 , 63 , 63
1330 DATA63 , 63 , 63 , 63 , 63 , 63 , 63 , 63
1340 DATA0 , 4 , 4 , 4 , 4 , 4 , 4 , 4
1350 RETURN
```

Chopper



You are the last survivor of a squadron of helicopters protecting your city. To complete your mission you must destroy all three enemy helicopters. Each of the enemy helicopters is better than the previous so the last battle is the true test. Both you and the enemy are equipped with high power lasers so destruction is instantaneous. The game is accompanied with the sound of the chopper's engine and laser fire.

Space bar = up
Left arrow = fire laser

The chopper drops by reducing lift, to go down you just let gravity take over. You can slow your downward descent by applying a little lift. Your chopper will land by itself if you let it.

STRUCTURE	LINE(S)
INITALIZATION	100-146
RESET GAME	150-199
MAIN LOOP	200-699
TURN ON COPTER SOUND	202
CLEAR LAST LASER FIRE FROM SCREEN	205, 206
MOVE ENEMY TO PLAYER'S HEIGHT	210
PUT ENEMY	220
PUT PLAYER	340
DECIDE IF ENEMY SHOULD FIRE	350
DISPLAY ENEMY LASER FIRE	360
TEST IF PLAYER HIT	370
GET KEYBOARD CHARACTER	400-440
INCREASE PLAYER'S UPWARD VELOCITY IF (SPACE) HIT	440
FIRE PLAYER'S LASER IF LEFT ARROW HIT	460
TEST IF ENEMY HIT	470
PUT ALTERNATE OF PLAYER'S COPTER	480
TEST IF A COPTER HAS MOVED INTO A LASER BEAM	485-487
GIVE PLAYER A DOWNWARD TERMINAL VELOCITY	510
ADJUST PLAYER VELOCITY	520
BOUNDARY PROTECTION	525-527
MOVE PLAYER	530
STOP PLAYER UPWARD MOMENTUM	540
END OF MAIN LOOP	699
'MISSION COMPLETE' SUBROUTINE	700-790
DATA	100-2030

VARIABLE	USE
P$()	PLATFORM CHARACTER STRINGS
C$()	ENEMY COPTER'S CHARACTER STRING
X()	ENEMY COPTER'S HORIZONTAL POSITIONS
Y()	ENEMY COPTER'S VERTICAL POSITIONS
PX	PLAYER'S HORIZONTAL POSITION
PY	PLAYER'S VERTICAL POSITION
MY	PLAYER'S OLD VERTICAL POSITION
YV	PLAYER'S VELOCITY INCREMENT
DV	PLAYER'S ACTUAL VELOCITY
CP	POINTER TO ENEMY IN FLIGHT
ZF	ENEMY'S LASER ON/OFF FLAG (0 OR 1)
F2	PLAYER'S LASER FLAG
L$	LASER BLAST STRING
L2$	LASER CLEAR STRING
ZP$	SECTION OF LASER STRING
CL$	SECTION OF LASER CLEAR STRING
X	HORIZONTAL POSITION OF ENEMY IN FLIGHT
Y	VERTICAL POSITION OF ENEMY IN FLIGHT
M	OLD VERTICAL POSITION OF ENEMY IN FLIGHT
C$	CHARACTER STRING OF ENEMY IN FLIGHT
A$	KEYBOARD CHARACTER

THE CHOPPERS

The choppers are made from predefined characters and may be altered by changing the decimal representation of the choppers in the DATA statements. Other characters may be redefined by inserting them with their redefinition into the DATA anywhere before the DATA "END" statement. All choppers were created using the graphics program. (As you can see there are 3 different types).

THE PLATFORMS

The platforms are constructed from the alternate character set though the program runs in text mode. This is achieved by copying the necessary alternate character formats into their standard character counterparts. The routine that does this resides in lines 112-114 in the program.

THE LASER BEAMS

The beam is a string of "-" characters that have been redefined to a double thickness full length version of the same character. A CHR$(1) (the code for Red) is added to the end of the laser string so as to leave the player's copter Red as the enemy or player fires. If this were not done then the player's chopper would change colour with the laser beam because of the setting of colour attributes. The length of the beam to be stored into ZP$ depends on the distance between the enemy and player. The colour of each beam is achieved by putting a colour code at the beginning of the beam.

THE CHOICE OF ENEMY

The same section of code is used for each enemy copter. This is achieved by storing the position and shape of each enemy copter into an array. The variable C is initially set to 1 and is incremented each time an enemy copter is shot down. When C reaches 3 and the enemy is shot down then the program jumps to 'mission complete'. Using arrays to keep track of moving objects also makes a program more flexible. If you wanted to include more copters in the program all you would have to do is narrow the landing pads, reposition the existing copters and allow the C variable to reach a higher value. Or if you wanted to have all the copters in the air at once you could 'cycle' the C variable inside the main loop (though this would slow down the game considerably). Using arrays as used in this game is advisable in most games.

```
 50 REM ********************
 52 REM *  C H O P P E R  *
 54 REM ********************
 55 REM
 57 REM
100 REM * INITIALIZATION *
110 CLS : POKE 618 , 10
111 PRINT "Please wait"
112 FOR D=1 TO 6 :  READ D$ : FOR I=0 TO 7
113 POKE46080+ASC(D$)*8+I , PEEK(47104+ASC(D$)*8+I) :
    NEXT I , D
114 DATA b , \ , # , 7 , U , £
120 C=C+1 : READ C$ : IF C$="END" THEN 140
125 V=46080+ASC(C$)*8
130 FOR I=V TO V+7 : READ D : POKE I , D
139 NEXT I : GOTO 120
140 FOR P=1 TO 3 : READ P$(P) : NEXT P
145 L$ ="----------------------------"+CHR$(1)
146 L2$="                              "+CHR$(1)+"  "
150 REM * START *
151 CLS : INK2 : PAPER0
157 FOR P=1 TO 3 : PLOT0 , P+23 ,
    CHR$(4)+P$(P)+P$(P)+P$(P)+"bbbbbb"+P$(P) : NEXT
160 FOR P=0 TO 22 : PLOT 24 , P , CHR$(1) : NEXT
170 FOR P=23 TO 15STEP-1 : PLOT 24 , P ,
    CHR$(4)+"bbbbbb"+CHR$(1) : NEXT
175 C$(1)="nopq" : C$(2)="nopq" : C$(3)="jklm"
178 X(1)=3 : X(2)=11 : X(3)=19 : Y=23 : Y(1)=Y :
    Y(2)=Y : Y(3)=Y : M(1)=Y : M(2)=Y : M(3)=Y
180 FOR K=1TO3 : PLOT X(K) , 23 , C$(K) : NEXT K
190 PX=33 : PY=23 : MY=5 : YV=.2 : DV=0 : C=0 :
    ZF=0 : F2=0
193 C=C+1 : IF C>3 THEN GOSUB 700 : GOTO150
194 PAPER 0
195 X=X(C) : Y=Y(C) : C$=C$(C) : J=(4-C)*2
197 ZP$=CHR$(7)+RIGHT$(L$ , 38-X-6) :
    CL$=CHR$(2)+RIGHT$(L2$ , 38-X-6)
199 ZAP : PLAY 0 , 0 , 0 , 0
200 REM * MAIN LOOP *
202 SOUND 1 , 7000 , 0 : PLAY 0 , 5 , 4 , 40
205 IF ZF=1THEN : ZAP : PLOTX+5 , Y , CL$ : ZF=0 :
    SOUND1 , 7000 , 0 : PLAY 0 , 5 , 4 , 40
206 IF F2=1 THEN PLOT 1 , MY , L2$ : F2=0
210 M=Y : Y=Y-SGN(INT(Y)-INT(PY))/((3-C)*2+1)
220 PLOT X , M , "  " : PLOT X , Y , C$
290 REM
300 REM
340 PLOTPX , MY , "  " : PLOTPX , PY , "acde"
350 IF Y>14 OR PY>14 OR RND(1)<.8 THEN 400
360 PLOT X+4 , Y , ZP$ : ZF=1
370 IF INT(Y)=INT(PY) THEN ZAP : EXPLODE :
    PAPER 1 : WAIT 100 : GOTO150
400 REM * GET A KEY *
420 PLAY 0 , 0 , 0 , 0 : PLAY 3 , 5 , 7 , 6
430 A$=KEY$ : IFA$=""THEN 480
440 IF A$=" "THEN DV=DV-YV*ABS(PY>3)5 : GOTO 480
```

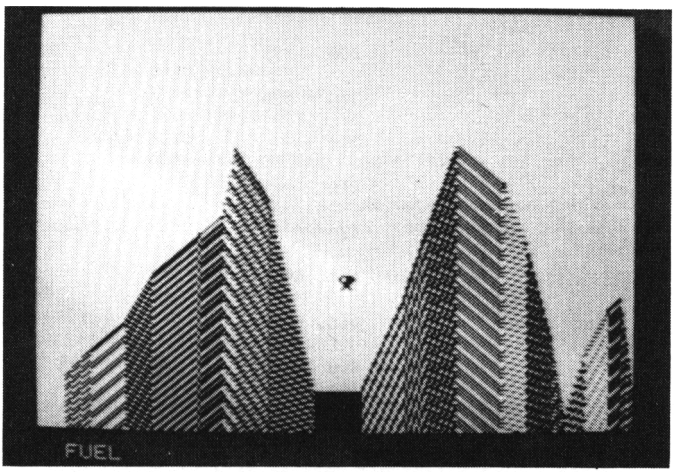
Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

(5#)

(4#)

```
445 IF PY>15 THEN 480
450 IF A$<>CHR$(8) THEN 480
460 PLAY 0 , 0 , 0 , 0 : ZAP : PLOT 1 , PY ,
    CHR$(4)+L$ : F2=1
470 IFINT(PY)=INT(Y)THEN PAPER 6 : EXPLODE : WAIT
    100 : PLOT 1 , Y , L2$ : GOTO 193
480 PLOT PX , PY , "fghi"
485 IF INT(Y)<>INT(PY)THEN490
486 IF ZF=1 THEN 370
487 IF F2=1 THEN 470
490 REM PLAY 0 , 0 , 0 , 0
500 REM MOVE PLAYER'S COPTER
510 IF DV<-1 THEN DV=-1
520 DV=DV+YV
525 IF PY+DV>24 AND DV> THEN DV=0
527 IF PY<3  AND DV<0 THEN DV=0
530 MY=PY : PY=PY+DV
540 IF DV<0 AND A$<>" " THEN DV=0
699 GOTO 200
700 REM MISSION COMPLETE
720 CLS : PAPER4 : INK7 : PRINT : PRINT : PRINT
730 POKE 48161 , 10 : POKE 48201 , 10
740 PRINTCHR$(4); : PRINT"  M I S S I O N
    C O M P L E T E !";CHR$(4)
780 WAIT 500
790 RETURN
1000 REM CHARACTER DATA
1010 DATA i , 0 , 9 , 2 , 62 , 4 , 9 , 0 , 0
1020 DATA h , 60 , 0 , 0 , 63 , 62 , 0 , 0 , 0
1030 DATA g , 63 , 16 , 56 , 23 , 27 , 23 , 62 , 60
1040 DATA f , 63 , 0 , 3 , 5 , 25 , 51 , 63 , 1
1045 REM
1050 DATA e , 0 , 0 , 0 , 42 , 0 , 0 , 0 , 0
1060 DATA d , 0 , 0 , 0 , 63 , 62 , 0 , 0 , 0
1070 DATA c , 0 , 16 , 56 , 31 , 31 , 31 , 62 , 60
1080 DATA a , 0 , 0 , 3 , 5 , 25 , 51 , 63 , 1
1085 REM
1100 REM ENEMY
1110 DATA j , 0 , 36 , 16 , 15 , 8 , 36 , 0 , 0
1120 DATA k , 31 , 0 , 0 , 63 , 63 , 1 , 0 , 0
1130 DATA l , 63 , 12 , 62 , 63 , 34 , 34 , 63 , 31
1140 DATA m , 62 , 0 , 0 , 48 , 8 , 6 , 63 , 62
1200 REM
1210 DATA n , 36 , 16 , 15 , 8 , 36 , 0 , 0 , 0
1220 DATA o , 7 , 0 , 63 , 31 , 7 , 3 , 1 , 0
1230 DATA p , 63 , 2 , 63 , 34 , 35 , 35 , 35 , 63
1240 DATA q , 63 , 0 , 48 , 8 , 6 , 63 , 63 , 62
1310 DATA - , 0 , 0 , 0 , 0 , 63 , 63 , 0 , 0
1990 DATA END
2000 REM PLATFORM
2010 DATA "\\\\\\\\"
2020 DATA " #7#7#7 "
2030 DATA " ###### "
3000 REM acde fghi onmk srqp
```

Luna Landa



Your module has been released from the mother ship orbiting around the moon.

The orbital velocity of the mother ship gives you forward momentum on your descent to the moon's surface.

When the game starts off you enter the picture at the point where you have to fire your rockets, to decrease your forward momentum, if you wish to avoid overshooting the landing pad.

You have a limited amount of fuel and so you must use your fuel wisely.

Any rocket you fire will first overcome any momentum working against it and then move the module. Once in motion the module has some momentum in that direction. You must skilfully use this momentum to your advantage or you will endanger the mission by using up your fuel.

Freefalling for as long as possible conserves fuel but you have to overcome the moon's gravity as well as your own momentum when you land and you must allow sufficient time for this.

Space bar is your upward thrust, left and right cursor keys are your manoeuvering rockets.

STRUCTURE :

FUNCTION	LINE(S)
Initialization	100 - 190
Draw mountain and platform	900 - 970
Game Reset	200 - 295
Main Loop	300 - 520
Cycle character	310
Get keyboard character	410
Jet sound	420
Amend velocity	470, 490
Calculate new position	460, 480
Test for collision	600 - 620
Put module	515
Landed routine	625 - 690
Crash routine	700 - 790
Data	1000 - 1070

VARIABLES :

K = General counters.
X = Modules Horizontal position
Y = Modules Vertical position
S = Vertical displacement
U = Vertical Velocity
A = Acceleration due to gravity
BX = Fuel burn in x direction
BY = Fuel burn in y direction
N = Character pointer
DIS = Character displacement
C() = Character array

HOW THE PROGRAM WORKS

Left arrow = left throttle
Right arrow = right throttle
Space bar = up thrust

Formulae :

All formulae used are 'straight-line' formulae that follow the laws of physics. The formulae used were :-

```
S = U * T + 0.5 * A * T**2
and
V = U + A * T
```

Where S = displacement
U = initial velocity
V = final velocity
T = time
A = acceleration due to gravity

After the time is set to 1 second intervals and the fuel burn B is subtracted from the acceleration, the formulae become :

```
S = U + 0.5 (A-B)
and
U = U + A-B
```

The Module :

The ASCII codes of the 12 module characters are stored into an array C(). The character displayed depends on 2 variables.

1) Variable-N which is continually cycled from 1 through 4 to create the effect of a light moving around the module.

2) Variable-DIS which is added to N to choose the rocket thrust character. The value of Dis (4 or 8) depends on the thrust key pressed.

Landscape :

After each crash or landing the Luna Landscape is redrawn. This is achieved by the following methods :

The cursor is first set to the bottom left-hand corner of the screen CURSET 0,199,3 and the variables x and y are used to keep track of the current cursor position. Random horizontal and vertical distances are then generated for each gradient of the mountain.

Rather than using a DRAW command to draw from the cursor's position to the bottom of the screen we used a random PATTERN command which adds some good effects to the moonscape. The position of the landing pad is fixed and for this we used Pattern 255 which is a solid block.

As each gradient is generated, it is tested against the variable 'P' which first contains the position of the platform and then the position of end of screen. If the gradient passes 'P' then the gradient is changed so as to line up exactly with the beginning of the platform.

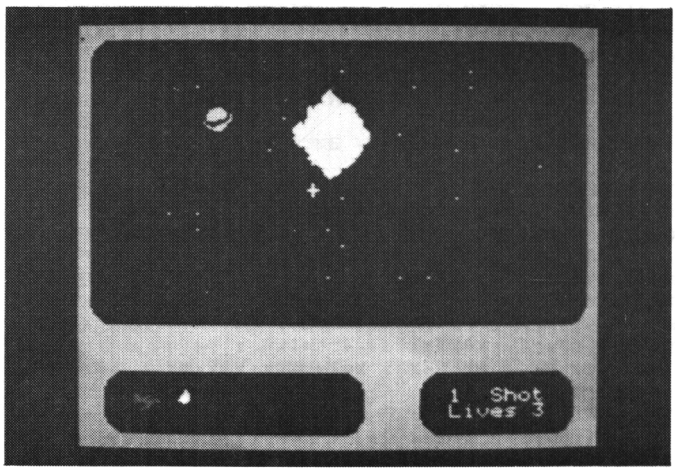
Also (in line 920) 5 has been added to the vertical component of the gradient to ensure that the slopes of the landscape cannot be mistaken for a landing pad.

If you wish to delete the forward momentum at the start change line 490 to X=X+BX*6.

```
100 REM LUNA LANDA (ORIC)
101 REM
105 DIM C(16) : REM CHARACTER ARRAY
110 POKE 618 , 10
123 PAPER4 : INK7 : HIRES : V=38912 : POKE 618 , 10
124 PRINT"PLEASE WAIT";
125 FOR K=1 TO 4
130 READ D$ : A=ASC(D$) : C(K)=A : C(K+4)=A+4 :
    C(K+8)=A+8 : C(K+12)=A+12
140 FOR I=0 TO 7
150 READ D
160 FOR K2=0 TO 3 : POKE V+A*8+K2*32+I , D : NEXT K2
170 NEXT I
180 NEXT K
185 FOR K=5 TO 13 STEP 4 : FOR I=4 TO 7 : READ B :
    FOR K2=K TO K+3
190 POKE V+C(K2)*8+I , B : NEXT K2 , I , K
200 REM
205 CLS : HIRES : PAPER 3 : INK 0 : POKE618 , 10
210 GOSUB 900 : REM MOUNTAIN
220 CLS : PRINT : PRINT"FUEL ";CHR$(27);CHR$(81);
230 POKE 616 , 1 : PRINT : POKE 617 , 35 :
    PRINTCHR$(27);CHR$(84);
290 D=50 : X=100 : Y=9 : H=9 : DIS=0 : A=6/D :
    B1=A*D/12 : B2=A*2 : U=0 : N=1 : F=35
291 XV=-B1*10  : U=5*A  : PF=1
295 CURSET X , Y , 3
299 REM
300 REM * START GAME *
301 REM
310 M=N+DIS : N=N+1 : IFN>4 THEN N=3
320 IF F<8 THEN K=0 : GOTO 440
330 F=F-.2 : POKE 616 , 1 : PRINT : POKE617 , F :
    PRINTCHR$(27);CHR$(84);
400 REM
410 K=PEEK(520)
420 IF K>131 THENPLAY 0 , 1 , 1 , 32767 : PF=1 :
    GOTO 445
440 IF PF=1THEN PF=0 : PLAY 0 , 0 , 0 , 0
445 BY=-(K=132)*B2
450 BX=((K=172)-(K=188))*B1
455 DIS=-(K=132)*4-(K=172)*8-(K=188)*12
460 S=U+(A-BY)/2
470 U=U+A-BY
480 H=H+S : Y=199-H
490 XV=XV+BX : X=X+XV
500 REM
505 IF X<12 OR X>232 OR Y<9 THEN CHAR C(M) , 0 , 0 :
    GOTO 700
507 CHARC(M) , 0 , 0
510 IFPOINT(X+6 , H+8)+POINT(X , H+8)<>0THEN600
515 CURSET X , 199-Y , 3 : CHAR C(N+DIS) , 0 , 1
520 GOTO 300
```

```
600 REM * COLLISION TEST *
620 IF H<PY-9 OR H>PY OR X<120-W/2 OR X>120-W/2+W OR
    U>6*A THEN 700
625 CURSET X , H , 3 : CHAR C(8) , 0 , 1
630 PRINT"LANDED";
690 PAPER6 : WAIT 800 : GOTO 200
700 REM CRASH ROUTINE
720 PRINT"CRASH !"; : EXPLODE
725 CURSET X , H , 3
730 FOR K2=1 TO 6 : FOR K=1TO12 : CHAR 95 , 1 , 0 :
    CHAR C(K) , 0 , 1 : PAPER 3
735 PAPER1 : NEXT K , K2
790 GOTO 200
900 REM MOUNTAIN ROUTINE
915 T=199-150 : Y=190 : X=0 : D=-1 : W=18 :
    P=INT(120-W/2) : PY=180
917 CURSETX , Y , 3
920 MY=INT(RND(1)*T+5)*D  : Y=Y+MY
923 PATTERN RND(1)*245+10
925 IF Y<T OR Y>199 THEN D=-D : Y=Y-MY*2 : MY=-MY
930 MX=INT(RND(1)*20)+5 : X=X+MX
935 IF X>=P THEN X=X-MX : MX=P-X : X=X+MX : Y=Y-MY :
    MY=PY-Y : Y=Y+MY
937 IF MX=0 THEN 930
939 X=X-MX : Y=Y-MY
940 K2=Y : FOR K=1 TO MX : K2=K2+MY/MX :
    CURSET X+K , K2 , 1 : DRAW 0 , 199-INT(K2) , 1 :
    NEXT
941 X=X+MX : Y=Y+MY : CURSET X , Y , 1
945 IF X=239 THEN RETURN
950 IF X<>P THEN 920
954 REM PLATFORM
955 P=239 : MX=W : X=X+W : MY=0 :PATTERN 255
970 GOTO 937
1000 REM DATA FOR EACH SHIP STAGE
1010 DATA a , 30 , 60 , 60 , 30 , 12 , 12 , 18 , 0
1020 DATA b , 30 , 15 , 15 , 30 , 12 , 12 , 18 , 0
1030 DATA c , 30 , 39 , 39 , 30 , 12 , 12 , 18 , 0
1040 DATA d , 30 , 57 , 57 , 30 , 12 , 12 , 18 , 0
1050 DATA 12 , 12 , 18 , 33
1060 DATA 8 , 12 , 31 , 0
1070 DATA 4 , 12 , 62 , 0
```

Meteor Storm



You are attempting to break the solo record flight from earth to Saturn. You are awakened by alarm bells on the last night. A message appears warning you that you are entering a heavy meteor storm. You can just make out Saturn to the left when the first meteor appears on the screen. The radar in the bottom left corner shows how close the meteor is. You must shoot the meteors before they ram your ship.

The controls for your laser cannons are: the cursor control keys and space bar to fire.

The cross-hairs show where your cannons are currently aimed.

HOW THE PROGRAM WORKS

When the cross-hairs are moved the character stored in V replaces the cross-hair. This is the character that was there originally. The character under the new cross-hair position is then stored in V so that when the cross-hair is moved again the character can replace the cross-hair. This ensures that the movement of the cross-hairs does not destroy the screen.

Whenever a meteor is removed from the screen (it has been hit or it has hit you) the complete display (Saturn, stars, cross-hairs) is redisplayed so that there are no gaps in the display, e.g. "behind" where the meteor was. Note that the star plot routine is purely random therefore the pattern of stars is different after each meteor. If you want the same pattern each time an array could be set up at the start of the game containing the positions of the stars then use that to redisplay the screen.

STRUCTURE	LINE(S)
INITIALIZE SCREEN	100-140
INITIALIZE CHAR SET	150-180
DEFINE ARRAY	200
INITIALIZE CONSOLE	210-530
INITIALIZE VARIABLES	540-620
MAIN LOOP	630-800
SOUND	810
IS CROSS-HAIR OVER A METEOR?	820-1030
EXPLODE METEOR	1030-1230
UPDATE RADAR	1240-1280
ROUTINES TO DRAW METEORS	1290-1570
COLLISION WITH METEOR	1580-1770
DATA	1800-1980

VARIABLES

X, Y = POSITION OF CROSS-HAIR
OX, OY = OLD POSITION OF CROSS-HAIR
L1 = NUMBER OF LIVES
MS = METEOR STAGE (0 - 9)
D1 = DIFFICULTY LEVEL
MX, MY = METEOR POSITION
V = WHAT IS UNDER THE CROSS-HAIR

```
100 CLS : PAPER0 : INK7
110 PLOT4 , 10 , CHR$(12)+"PREPARE FOR FLIGHT INTO"
120 PLOT10 , 12 , CHR$(12)+"METEOR STORM"
130 SOUND1 , 10000 , 0 : MUSIC1 , 2 , 1 , 0 :
    PLAY1 , 0 , 3 , 1300
140 PRINT CHR$(20)CHR$(6)CHR$(17)
150 FORI=47360 TO 47375 : POKEI , 0 : NEXT I :
    POKE47375 , 4
160 FORI=47624 TO 47624+38*8
170 :  READ X : POKE I , X
180 NEXTI
190 REM
200 DIM S(10) : CLS : PLAY0 , 0 , 0 , 0
210 FORI=0TO26
220 :  PLOT0 , I , CHR$(9)
230 NEXTI
240 PLOT2 , 0 ,
    "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
250 PLOT37 , 1 , "B"
260 FORI=1TO18
270 :  PLOT2 , I , "e" :  PLOT38 ,RI , "e"
280 NEXTI
290 PLOT3 , 18 , "C" : PLOT37 , 18 , "D"
300 PLOT2 , 19 ,R
    "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
310 PLOT2 , 20 ,
    "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
320 PLOT2 , 21 ,
    "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
330 PLOT2 , 22 , "eeA                Bee
    eeA          ee"
340 PLOT2 , 23 , "ee  EF              ee
    ee      ee"
350 LI$="ee  GH      eeee "+CHR$(8)+"Lives
    3"+CHR$(9)+" ee"
360 PLOT2 , 24 , LI$
370 PLOT2 , 25 , "eeC              Dee
    eeC     Dee"
380 PLOT2 , 26 ,
    "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee"
390 FORI=0TO26
400  :   PLOT1 , I , CHR$(1)
410 IF I>0 AND I<19 THEN PLOT 3 , I , CHR$(3) :
    PLOT 36 , I , CHR$(1)
420 NEXT I : PLOT 3 , 1 , "A" : PLOT3 , 18 , "C"
430 PLOT4 , 23 , CHR$(4) : PLOT8 , 23 , CHR$(3) :
    PLOT21 , 23 , CHR$(1)
440 PLOT4 , 24 , CHR$(4) : PLOT8 , 24 , CHR$(1)
450 PLOT26 , 24 , CHR$(2) : PLOT36 , 24 , CHR$(1)
460 PLOT26 , 23 , CHR$(2) : PLOT36 , 23 , CHR$(1)
470 GOSUB480 : GOTO540
480 FORI=1TO20
490 :  PLOT4+32*RND(1) , 1+16*RND(1) , "!"
```

(16#) (14#) (11#) (14#) (16#) (9#)

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

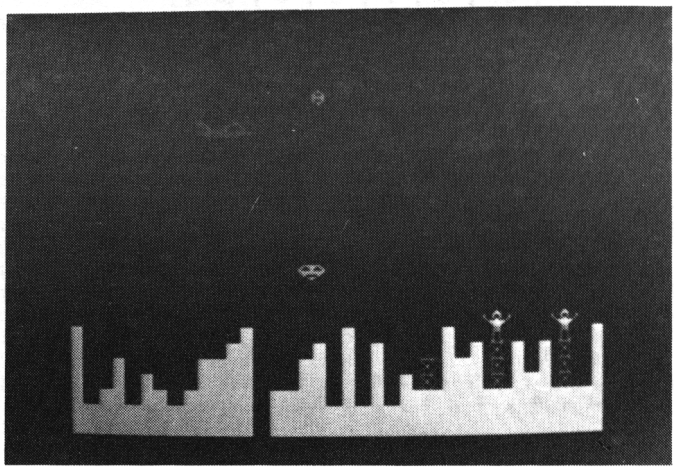
```
500 NEXT I
510 PLOT10 , 5 , CHR$(5)+"IJ"+CHR$(3)
520 PLOT10 , 6 , CHR$(5)+"KL"+CHR$(3)
530 RETURN
540 OX=9 : OY=15 : OV=32
550 X=9 : Y=15 : V=32
560 PO=48000+(X*40)+Y : POKEPO , 102
570 LI=3 : DI=1 : NM=0
580 MS=0 : NM=NM+1
590 MX=INT(4+RND(1)*12)
600 MY=INT(7+RND(1)*25)
610 MI=1
620 PO=48000+(X*40)+Y : POKEPO , 102
630 A=PEEK(520)
640 MS=MS+DI/7+NM/50
650 IF INT(MS)=9 THEN GOTO 1580
660 GOSUB 1240
670 GOSUB 1290+100*INT(S)
680 IF A<>188 AND A<>156 AND A<>172 AND A<>180 THEN
    GOTO 800
690 IF Y>9 AND Y<13 AND X>5 AND X<8 THEN GOSUB 510
700 Y=OY+(A=172)-(A=188)
710 IF Y<6 THEN Y=35
720 IF Y>35 THEN Y=6
730 X=OX+(A=156)-(A=180)
740 IF X<3 THEN X=18
750 IF X>18 THEN X=3
760 PO=48000+(X*40)+Y : V=PEEK(PO)
770 PO=48000+(OX*40)+OY : POKEPO , OV
780 PO=48000+(X*40)+Y : POKEPO , 102
790 OX=X : OY=Y : OV=V
800 IF A<>132 THEN GOTO 630
810 ZAP
820 IF MS<1 THEN GOTO 630
830 IF MS>2 THEN GOTO 850
840 IF MX<>X OR MY<>Y THEN GOTO 630 ELSE GOTO 1030
850 IF MS>3 THEN GOTO 870
860 IF (MY=YORMY+1=Y) AND MX=X THEN GOTO 1030 ELSE
    GOTO 630
870 IF MS>4 THEN GOTO 890
880 IF ((MY=YORMY+1=Y) AND MX=X) OR
    (MX+1=X AND MY=Y) THEN GOTO 1030 ELSE GOTO 630
890 IF MS>5 THEN GOTO 910
900 IF MY=Y AND (MX=XORMX=MX+1) THEN GOTO 1030 ELSE
    GOTO 630
910 IF MS>7 THEN GOTO 930
920 IF (MX=X OR MX+1=X) AND (MY=YORMY+1=Y)
    THEN GOTO 1030 ELSE GOTO 630
930 IF MS>8 THEN GOTO 980
940 IF (X=MX-1 AND (Y=MYORY=MY+1)) THEN GOTO 1030
950 IF (X=MXORX=MX+1) AND (Y>=MY-1 AND Y<=MY+2)
    THEN GOTO 1030
960 IF X=MX+2 AND (Y=MY OR Y=MY+1) THEN GOTO 1030
970 GOTO 630
980 IF MS>10 THEN GOTO 630
```

```
 990 IF (X=MX-2 OR X=MX+3) AND (Y=MY OR Y=MY+1)
     THEN GOTO 1030
1000 IF (X=MX-1 OR X=MX+2) AND (Y>=MY-1 AND Y<+MY+2)
     THEN GOTO 1030
1010 IF (X=MX OR X=MX+1) AND (Y>=MY-2 AND Y<=MY+3)
     THEN GOTO 1030
1020 GOTO 630
1030 X1=X-2 : IF X1<0 THEN X1=0
1040 Y1=Y-2 : IF Y1<0 THEN Y1=0
1050 X2=X-1 : IF X2<0 THEN X2=0
1060 EXPLODE
1070 PLOTY , X , "£"
1080 PLOTY1 , X1 , CHR$(96)+"ab"
1090 PLOTY1 , X , "b£b"
1100 PLOTY1 , X2 , "cdc"
1110 PAPERRND(1)*7
1120 FORI=1TO10
1130 :  PAPERRND(1)*7
1140 NEXTI :PAPER0
1150 FORI=1TO18
1160 :   PLOT4 ,
     I , "                                    "
1170 NEXTI
1180 PLOT9 , 23 , "                 "
1190 OV=32 : V=32
1200 PLOT26 , 23 , CHR$(8)+CHR$(2) : PLOT27 , 23 ,
     STR$(NM+LI-3)
1210 PLOT31 , 23 , "Shot"+CHR$(1)+CHR$(9)
1220 GOSUB480
1230 GOTO580
1240 IF INT(MS)<4 THEN PLOT 19-MS , 23 , " N "
1250 IF INT(MS)=4 THEN PLOT 19-MS , 23 , "  N "
1260 IF INT(MS)>4 AND INT(MS)<8 THEN
     PLOT 23-2*INT(MS) , 23 , "  N  "
1270 IF INT(MS)=8 THEN PLOT9 , 23 , "N  "
1280 RETURN
1290 REM  METEOR I
1300 PLOT MY-1 , MX-1 , "M"
1310 RETURN
1320 REM  METEOR II
1330 PLOT MY-1 , MX-1 , "N"
1340 REM  METEOR III
1350 PLOT MY-1 , MX-1 , "NR"
1360 RETURN
1370 REM  METEOR IV
1380 PLOT MY-1 , MX-1 , "NR" : PLOTMY-1 , MX , "P"
1390 RETURN
1400 REM  METEOR V
1410 PLOT MY-1 , MX-1 , "O" : PLOTMY-1 , MX , "Q"
1420 RETURN
1430 REM  METEOR VI
1440 PLOT MY-1 , MX-1 , "OR" : PLOTMY-1 , MX , "QS"
1450 RETURN
1460 REM  METEOR VII
1470 PLOT MY-1 , MX-1 , "TU" : PLOTMY-1 , MX , "VW"
1480 RETURN
```

```
1490 REM  METEOR VIII
1500 PLOT MY-1 , MX-2 , "OR" : PLOTMY-2 , MX-1 ,
     "OeeR"
1510 PLOT MY-2 , MX , "QeeS" : PLOTMY-1 , MX+1 , "QS"
1520 RETURN
1530 REM  METEOR IX
1540 PLOT MY-1 , MX-3 , "OR" : PLOTMY-2 , MX-2 ,
     "TeeU"
1550 PLOT MY-3 , MX-1 , "OeeeeR" : PLOTMY-3 , MX ,
     "QeeeeS"
1560 PLOT MY-2 , MX+1 , "VeeW" : PLOTMY-1 , MX+2 ,
     "QS"
1570 RETURN
1580 REM YOU ARE HIT
1590 FORI=1TO100
1600 :  IF B=0 THEN B=7
1610 :  PAPERB
1620 :  EXPLODE
1630 NEXTI
1640 PAPER0
1650 LI=LI-1
1660 IF LI>0 THEN GOTO 1760
1670 SOUND5 , 30000 , 0 : MUSIC1 , 2 , 1 , 0 :
     PLAY7 , 0 , 1 , 2500
1680 WAIT75
1690 MUSIC1 , 1 , 1 , 0 : PLAY7 , 0 , 1 , 2500
1700 CLS
1710 PRINT CHR$(6)CHR$(17)CHR$(20)
1720 PLAY0 , 0 , 0 , 0
1730 INK2
1740 PLOT10 , 10 ,
     "You shot"+STR$(NM+LI-3)+" meteors !"
1750 END
1760 PLOT27 , 24 ,
     CHR$(8)+"Lives"+STR$(LI)+CHR$(9)+CHR$(1)
1770 GOTO1150
1780 END
1790 FORI=0TO26 : PLOT0 , I , CHR$(8) : NEXT :
     PRINT CHR$(17)CHR$(6)
1800 DATA63 , 62 , 60 , 56 , 48 , 32 , 32 , 32 , 63 ,
     31 , 15 , 7 , 3 , 1 , 1 , 1 , 32 , 32 , 32 ,
     48 , 56 , 60 , 62 , 63
1810 DATA1 , 1 , 3 , 3 , 7 , 7 , 15 , 63
1820 DATA0 , 0 , 32 , 4 , 56 , 63 , 63 , 63 , 0 ,
     0 , 0 , 0 , 48 , 36 , 30 , 63 , 1 , 1 , 3 , 6 ,
     0 , 0 , 0 , 0 , 56 , 48 , 32
1830 DATA0 , 0 , 0 , 0 , 0
1840 DATA0 , 0 , 3 , 15 , 15 , 31 , 31 , 31 , 0 , 0 ,
     48 , 56 , 60 , 61 , 61 , 57
1850 DATA31 , 32 , 32 , 25 , 15 , 3 , 0 , 0 , 3 , 6 ,
     28 , 60 , 56 , 48 , 0 , 0
1860 DATA0 , 0 , 1 , 1 , 3 , 7 , 7 , 2 , 0 , 2 , 2 ,
     7 , 15 , 15 , 31 , 6
1870 DATA0 , 2 , 6 , 15 , 31 , 31 , 31 , 15 , 14 ,
     14 , 12 , 4 , 0 , 0 , 0 , 0
```

```
1880 DATA15 , 31 , 15 , 15 , 7 , 2 , 0 , 0 , 0 ,
     0 , 0 , 32 , 48 , 56 , 56 , 56 , 56 , 48 ,
     0 , 0 , 0 , 0
1890 DATA1 , 11 , 27 , 31 , 31 , 31 , 31 , 32 ,
     48 , 56 , 56 , 48 , 60 , 60 , 62
1900 DATA31 , 15 , 15 , 31 , 5 , 7 , 5 , 0 , 62 ,
     62 , 60 , 48 , 56 , 48 , 32 , 0
1910 DATA3 , 31 , 63 , 31 , 63 , 63 , 31 , 15 , 62 ,
     60 , 62 , 63 , 63 , 60 , 60 , 62
1920 DATA1 , 11 , 11 , 15 , 31 , 31 , 31 , 31 , 32 ,
     48 , 56 , 56 , 48 , 60 , 60 , 62 , 31 , 15 ,
     15 , 31 , 15 , 15 , 5
1930 DATA33 , 55 , 30 , 18 , 18 , 30 , 56 , 33 , 33 ,
     19 , 19 , 30 , 30 , 42 , 51 , 51
1940 DATA45 , 45
1950 DATA18 , 14 , 58 , 62 , 22 , 19 , 49 , 30 , 27 ,
     27 , 58 , 42 , 14 , 46 , 58 , 17 , 46 , 26 ,
     29 , 54 , 44 , 25
1960 DATA63 , 32
1970 DATA18 , 14 , 58 , 62 , 22 , 19 , 49 , 30 , 27 ,
     27 , 58 , 42 , 14 , 46 , 58 , 17 , 46 , 26 ,
     29 , 54 , 44 , 25
1980 DATA63 , 63 , 63 , 63 , 63 , 63 , 63 , 63 , 63 ,
     4 , 4 , 4 , 31 , 4 , 4 , 4 , 0 , 0
```

Eliminator



You are the protector and defender of the humanoids on the surface of the planetoid. Your task is to ELIMINATE the invader ALIENS.

You must prevent at all costs the alien's attempts to kidnap the humanoids under your care. Their survival and the survival of their planetoid depends on you.

Because you have set your craft to battle mode it automatically maintains a height minimum. Thus you cannot fly too low. Your speed is also set to maximum allowing you to concentrate fully on blasting the aliens out of the sky.

The enemy kidnappers have sufficient intelligence to evade your only weapon, the horizontal heat ray. You must not allow them to descend below your lowest possible flight path or you will not be able to stop the invasion and thus lose your humanoids.

HOW TO PLAY THE GAME

You control the space ship on the screen; your forward momentum does not allow you any flexibility in moving sideways. Your mission is to save the humanoids from the Alien. You may have 1, 2 or 3 humanoids to protect. You have a heat ray at your disposal to help you in your duty, although this brings only temporary relief. A smart bomb is also there to make your job harder.

Use "cursor up" to move up
"cursor down" to move down
"SPACE BAR" key to fire heat ray.

Eliminator uses a machine code program to simulate flying over the surface of the planet. This program is 44 bytes long and sits in the data lines from line 3000. During initialization, the program is POKEd high up in Oric's memory. H8000, (32768). The program is fully relocatable and may be placed anywhere convenient in memory.

The program simply takes the last 8 lines of the screen and moves each character one to the left i.e. Character at (0,1) will goto (0,0) and character at (0,0) will goto (0,38), etc.

STRUCTURE	LINE(S)
INITIALIZE UDG, DRAW LAND, PLACE TOWERS AND MEN, GET INITIAL CO-ORDINATES FOR SHIP - ALIEN AND BOMB. POKE M/C INTO MEMORY	5-360
DRAW SHIP, ALIEN AND BOMB, M/C ROUTINE	1000-1020
MAIN LOOP - CHECK FOR SHIP MOVEMENT MAKE BOMB HEAD FOR SHIP, CHECK IF SHIP IS FIRING AT ALIEN OR BOMB, FIRE LASER IF SO. CHECK FOR POSSIBLE COLLISIONS.	5000-5360
THIS IS THE KIDNAP LOOP. CHECK IF ANY HUMANS LEFT, ETC.	7000-7330
DAMAGE TO SHIP ROUTINE INCLUDES END OF PROGRAM.	8500-8760
SUBROUTINE WHICH DRAWS LAND AND PLACES TOWERS AND HUMANS ON IT.	9000-9060

SCROLL PROGRAM DISASSEMBLY

```
A2 08              LDX    #8
A9 A0              LDA    STARTH
85 40              STA    POINT
A9 BE              LDA    STARTL
85 41              STA    POINT+1

A0 01    START:    LDY    #1
B1 40              LDA    (POINT),Y
48                 PHA

C8       NEXTCH:   INY
B1 40              LDA    (POINT),Y
88                 DEY
91 40              STA    (POINT),Y
C8                 INY
C0 27              CPY    #39
D0 F5              BNE    NEXTCH

68                 PLA
91 40              STA    (POINT),Y

18                 CLC
A9 28              LDA    #40
65 40              ADC    POINT
85 40              STA    POINT
90 02              BCC    AGAIN
E6 41              INC    POINTH
CA       AGAIN:    DEX
D0 DF              BNE    START

60                 RTS
```

```
  3 REM * ELIMINATOR *
  4 REM
  5 CLS : PAPER4 : INK7 : N=0
  6 PRINT CHR$(17);CHR$(6)
  7 POKE 618 , 10
 10 UP$=CHR$(11) : DOWN$=CHR$(10) : FIRE$=CHR$(9)
100 RESTORE
110 FOR J=65 TO 77
120 FOR K=0 TO 7
130 READ X
140 POKE 46080+(J*8)+K , X
150 NEXT K
160 NEXT J
170 FOR J=0 TO 43
180 READ K
190 POKE 32768+J , K
200 NEXT J
210 GOSUB 9000
310 SX=10 : SY=15
320 AY=INT(RND(1)*15) : AX=34
330 BY=INT(RND(1)*15) : BX=30 : IF AY=BY THEN 330
340 T=0
350 E=0 : H=3
360 DM=0 : SC=0
1000 PLOT SX , SY , "GHIJ"
1001 PLOT SX-1 , SY , CHR$(5) : PLOT SX+4 , SY ,
     CHR$(4)
1010 PLOT AX , AY , "KL"
1011 PLOT AX-1 , AY , CHR$(6) : PLOT AX+2 , AY ,
     CHR$(4)
1020 PLOT BX , BY , "M"
1021 PLOT BX-1 , BY , CHR$(2) : PLOT BX+1 , BY ,
     CHR$(4)
2000 REM PLATFORM
2010 DATA 7 , 4 , 2 , 1 , 1 , 2 , 4 , 2
2020 DATA 56 , 8 , 16 , 32 , 32 , 16 , 8 , 16
2030 REM SHAPE OF HUMANS
2040 DATA 1 , 3 , 34 , 34 , 17 , 15 , 7 , 3
2050 DATA 32 , 48 , 17 , 17 , 34 , 60 , 56 , 48
2060 DATA 3 , 3 , 3 , 2 , 2 , 2 , 2 , 6
2070 DATA 48 , 48 , 48 , 16 , 16 , 16 , 16 , 24
2080 REM SPACE SHIP
2090 DATA 60 , 62 , 31 , 15 , 7 , 63 , 63 , 7
2100 DATA 0 , 0 , 1 , 35 , 63 , 63 , 63 , 63
2110 DATA 7 , 31 , 56 , 50 , 49 , 63 , 63 , 63
2120 DATA 0 , 32 , 48 , 24 , 12 , 63 , 63 , 0
2130 REM ALIEN
2140 DATA 15 , 27 , 49 , 63 , 31 , 12 , 3 , 1
2150 DATA 60 , 54 , 35 , 63 , 62 , 12 , 48 , 32
2160 REM SMART BOMB
2170 DATA 12 , 30 , 51 , 63 , 63 , 51 , 30 , 12
3000 DATA 162 , 8 , 169 , 160 , 133 , 64 , 169 ,
     190 , 133 , 65
```

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: 12#.

```
3010 DATA 160 , 2 , 177 , 64 , 72
3020 DATA 200 , 177 , 64 , 136 , 145 , 64 , 200 ,
     192 , 39 , 208 , 245
3030 DATA 104
3040 DATA 145 , 64
3050 DATA 24 , 169 , 40 , 101 , 64 , 133 , 64 , 144 ,
     2 , 230 , 65
3060 DATA 202 , 208 , 223 , 96
5000 CALL 32768
5010 X$=KEY$
5020 NY=SY+(X$=DOWN$)*(SY<18)-(X$=UP$)*(SY>0)
5030 IF NY=SY THEN 5060
5040 V$=CHR$(SCRN(SX+1 , NY)) : W$=CHR$(SCRN(SX+2 ,
     NY)) : X$=CHR$(SCRN(SX+3 , NY))
5042 V$=V$+W$+X$
5050 IF V$<>"    " THEN 8500
5060 IF SCRN(SX+5 , NY)<>32 THEN 8500
5070 PLOT SX-1 , SY , "       "
5075 PLOT SX , NY , "GHIJ"
5076 PLOT SX-1 , NY , CHR$(5) : PLOT SX+4 ,
     NY , CHR$(4)
5080 SY=NY
5090 BX=BX-1 : IF BX<2 THEN PLOTBX+1 , BY , " ↓ " :
     BX=32                                          (3#)
5100 IF RND(1)>0.7 OR BX=32 THEN 5140
5110 NY=BY-(SY-1>BY)+(SY-1<BY)+(BX=SX+2)
5120 PLOT BX+1 , BY , " ↑  "                        (4#)
5125 PLOT BX , NY , "M"
5126 PLOT BX-1 , NY , CHR$(2) : PLOT BX+1 , NY ,
     CHR$(4)
5130 BY=NY
5140 IF BY=SY AND ABS(BX-SX-1)<=1 THEN 8500
5200 IF X$<>FIRE$ THEN 5290
5210 ZAP
5220 PLOT SX+4 , SY , 17
5240 ZAP
5250 PLOT SX+4 , SY , 20
5270 IF INT(AY)=INT(SY) AND AX>SX+4 AND AX<37 THEN GOSUB 8000
5280 IF BY=SY AND BX>SX+4 AND BX<37  THEN GOSUB 8100
5290 AX=AX-1 : IF AX<2 THEN PLOT AX+1 , AY , " ↑  " :
     AX=32                                          (4#)
5300 IF RND(1)>0.7 OR AX>=32  THEN 5000
5310 NY=AY-(AY<18)-T : IF AY=18 THEN 5330
5320 V$=CHR$(SCRN(AX+1 , NY)) : W$=CHR$(SCRN(AX+2 ,
     NY)) : IF V$+W$<>"  " THEN 7000
5330 NX=AX-(AY=18)
5340 IF NX<>AX AND SCRN(NX+2 , NY)<>32 THEN 7000
5350 PLOT AX+1 , AY , "  ↑  "                       (6#)
5535 PLOT NX , NY , "KL"
5356 PLOT NX-1 , NY , CHR$(6) : PLOT NX+2 , NY ,
     CHR$(4)
5360 AX=NX : AY=NY
6000 GOTO 5000
7000 IF ABS(BY-AY)<=1 AND ABS(BX-AX)<=1 THEN EXPLODE
     : REM ALIEN , BOMB CRASH
7010 IF ABS(SY-AY)<=2 AND ABS(BX-SX)<=2 THEN 8500
```

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

```
7020 PLOT AX , AY , "  ↑   "                                    (5#)
7025 PLOT AX , AY-1 , "KL"
7026 PLOT AX-1 , AY-1 , CHR$(6) : PLOT AX+2 , AY-1 ,
     CHR$(4)
7030 AY=AY-1 : AX=AX-1 : BX=BX-1
7040 IF AX<2 THEN PLOT AX+1 , AY+1 , "  ↓    " : AX=32          (6#)
7050 IF BX<2 THEN PLOT BX+1 , BY , " ↑ " : BX=32                (3#)
7055 PLOT BX+1 , BY , " ↓ "
7056 PLOT BX , BY , "M"
7057 PLOT BX-1 , BY , CHR$(2) : PLOT BX+1 , BY ,
     CHR$(4)
7060 CALL 32768
7070 PLOT SX-1 , SY , "  ↓    "                                 (6#)
7075 PLOT SX , SY , "GHIJ"
7076 PLOT SX-1 , SY , CHR$(5) : PLOT SX+4 , SY ,
     CHR$(4)
7080 F=0
7090 FOR Y=AY TO AY+4
7095 IF Y>23 THEN 7125
7100 FOR X=AX-2 TO AX+2
7105 IF X<2 OR X>36 THEN 7120
7110 IF CHR$(SCRN(X , Y))="D"OR CHR$(SCRN(X , Y))="C"
     THEN PLOTX , Y , "  " : F=F+1
7115 IF CHR$(SCRN(X , Y))="E" OR CHR$(SCRN(X ,
     Y))="F" THEN PLOTX , Y , "  " : F=F+1
7120 NEXT X
7125 NEXT Y
7130 IF F<>OTHEN 7200
7145 AX=AX+1 : PLOT AX-1 , AY , " ↓ "                           (3#)
7146 PLOT AX , AY , "KL"
7147 PLOT AX-1 , AY , CHR$(6) : PLOT AX+2 , AY ,
     CHR$(4)
7160 GOTO 5000
7200 PLOT AX , AY , " ↓  "                                      (4#)
7205 PLOT AX , AY-1 , "KL"
7206 PLOT AX-1 , AY , CHR$(6) : PLOT AX+2 , AY ,
     CHR$(4)
7210 AY=AY-1 : AX=AX-1
7215 IF AX<2 THEN PLOT AX+1 , AY+1 , " ↓  " : AX=32             (4#)
7220 BX=BX-1 : IF BX<2 THEN PLOT BX+1 , BY , "    " :
     BX=32
7230 CALL 32768
7240 PLOT AX , AY+1 , "CD"  : PLOT AX , AY+2 , "EF"
7250 FOR K=AY-1 TO 0 STEP -1
7260 PLOT AX , K+1 , "KL"
7261 PLOT AX-1 , K+1 , CHR$(6) : PLOT AX+2 , K+1 ,
     CHR$(4)
7262 PLOT AX , K+2 , "CD"
7264 PLOT AX , K+3 , "EF"
7265 PLOT AX , K , "KL" : PLOT AX-1 , K , CHR$(6) :
     PLOT AX+2 , K , CHR$(4)
7266 PLOT AX , K+1 , "CD"
7267 PLOT AX , K+2 , "EF"
7268 PLOT AX , K+3 , "    "
7270 PING
7280 NEXT K
```

```
7290 N=N-1
7300 PLOT SX-1 , SY , "  "
7305 PLOT SX , SY , "GHIJ"
7306 PLOT SX-1 ,
     SY , CHR$(5) : PLOT SX+4 , SY , CHR$(4)
7310 PLOT AX , 0 , "    " : PLOT AX , 1 , "    " :
     PLOT AX , 2 , "    "
7320 AX=AX-1 : IF AX<2 THEN PLOT AX+1 , AY , "  " :
     AX=32
7325 PLOT AX , AY , "  " : AY=6
7326 IF N=0 THEN 8550
7330 GOTO 5000
8000 EXPLODE : FOR K=1 TO 20
8010 PLOT AX , AY , "KL"
8012 PLOT AX-1 , AY , CHR$(1) : PLOT AX+2 , AY ,
     CHR$(4)
8015 PLOT AX , AY , "LK"
8016 WAIT(5)
8020 PLOT AX , AY , "  "
8030 NEXT K
8040 AY=(RND(1)*15) : AX=29
8050 PLOT AX , AY , "KL"
8051 PLOT AX-1 , AY , CHR$(6) : PLOT AX+2 , AY ,
     CHR$(4)
8060 SC=SC+200
8070 RETURN
8100 EXPLODE : FOR K=1 TO 20
8110 PLOT BX , BY , "M"
8112 PLOT BX-1 , BY , CHR$(1) : PLOT BX+1 , BY ,
     CHR$(4)
8115 WAIT(10)
8120 PLOT BX , BY , "  "
8130 NEXT K
8140 BY=(RND(1)*14) : BX=30
8150 PLOTBX , BY , "M"
8151 PLOTBX-1 , BY , CHR$(2) : PLOT BX+1 , BY ,
     CHR$(4)
8160 SC=SC+20
8170 RETURN
8500 PLOT 22 , 4 , "damage to ship."
8505 PLOT BX , BY , "   " : PLOT AX , AY , "  "
8510 DM=DM+1 : IF DM<3 THEN 8700
8520 PLOT 22 , 5  , "irrepairable."
8550 REM
8570 PLOT15 , 7 , "life support systems"
8571 PLOT15 , 8 , "for humans no longer"
8572 PLOT15 , 9 , "functioning.you have"
8573 PLOT15 , 10 , "failed to protect"
8574 PLOT15 , 11 , "your planet."
8650 PRINT"your score "SC
8655 WAIT(100)
8660 PRINT"do you want another game , y/n" : INPUT A$
     : IF A$="Y" THENRUN
8670 END
8700 EXPLODE : PLOT SX-1 , SY , CHR$(1) : PLOT SX+4 ,
     SY , CHR$(4) : FOR K=1 TO 20
```

5#

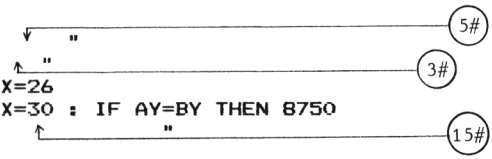
4#

5#

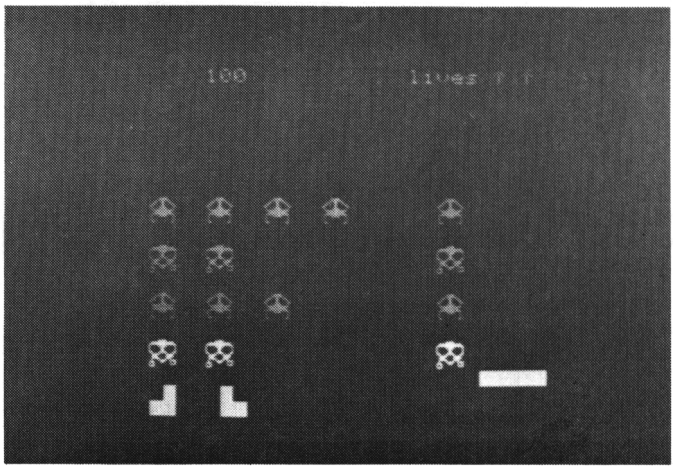
4#

3#

```
8710 PLOT SX , SY , "GHIJ"
8712 PLOT SX , SY , "HGIJ"
8714 WAIT(5)
8715 NEXT K
8730 PLOT AX-1 , AY , "  "
8735 PLOT BX-1 , BY , "  "
8740 AY=(RND(1)*15) : AX=26
8750 BY=(RND(1)*15) : BX=30 : IF AY=BY THEN 8750
8760 PLOT 22 , 4 , "          "
8770 GOTO 1000
9000 FOR X=2 TO 38
9010 FOR Y=26 TO 19+(RND(1)*6) STEP-1
9020 PLOT X , Y , 130
9030 NEXT Y
9050 NEXT X
9060 FOR X=2 TO 38 STEP 5
9070 IF SCRN(X , 23)<>32 AND SCRN(X+1 , 23)<>32 AND
     N<3 THEN GOSUB 9200
9090 NEXT X
9100 RETURN
9200 N=N+1 : FOR I=0 TO (RND(1)*2)+1
9210 PLOT X , 23-I , "AB"
9220 NEXT I
9230 PLOT X , 23-I , "EF"
9240 PLOT X , 23-I-1 , "CD"
9250 RETURN
```



ORIC Invaders



This is it, the all time favourite arcade game. The game that started it all and is still one of the most popular around the world, and it is right here on your ORIC.

There are 24 animated invaders and you have only 3 lives. You have the job of defending the planet against the invading intruders.

Your controls are: "Space bar" to fire, left and right cursor to move your laser.

HOW THE PROGRAM WORKS

It would be almost impossible to write this sort of game keeping track of the co-ordinates of each of 24 individual Invaders let alone trying to maintain some speed while doing it.

The way it is done is to let the program rely on a series of strings which contain one complete line of Invaders each. Doing it this way allows a small number of variables to be used and only a few strings need to be printed each cycle. The end result is that we can maintain control and still retain speed.

There are two "sets" of strings which are almost identical (to contain invaders in the same positions etc.) but have a different shaped Invader. By alternating these two "sets" the Invaders appear to be animated. To alternate between the strings as fast as possible both "sets" are printed in one cycle with all the processing being duplicated for each set. This method saves time by not having variables and "IF" statements with the printing of the Invaders.

When you hit an Invader with one of your missiles his place in the strings is taken up by a group of blanks and a check is made to see if there are any live Invaders left on the line. Lines which don't have any Invaders on them are not printed. This is what produces the effect of the Invaders coming down when they have the room to do so, and takes away the risk of pushing characters off the bottom of the screen. Because less lines are printed it takes less time to draw the Invaders and so the game speeds up towards the end.

STRUCTURE	LINE(S)
INITIALIZE SCREEN AND VARIABLES	100-230
INITIALIZE STRINGS THAT CONTAIN INVADERS	240-340
DRAW BARRIERS	360-390
DRAW LASER BASE	400
MOVE INVADERS	410-420
DRAW INVADERS	490-590
ARRIVED? YES, END GAME	600
MOVE AND DRAW LASER BASE	610-670
FIRE? YES, INITIALIZE MISSILE CO-ORDINATES	680-700
IF MISSILE ON SCREEN MOVE IT AND CHECK FOR IT HITTING ANYTHING	710-750
IF INVADER MISSILE ON SCREEN MOVE IT AND CHECK FOR IT HITTING ANYTHING	760-820
IF NO INVADER MISSILE ON SCREEN INITIALIZE A NEW ONE	830-860
SECOND PART OF CYCLE - SAME AS 1000-1730 BUT PRINTING DIFFERENT STRINGS (SEE HOW PROG WORKS)	870-1260
MISSILE HIT SOMETHING - REMOVE FROM SCREEN	1270-1570
INVADER MISSILE HIT SOMETHING - REMOVE FROM SCREEN	1580-1610
END OF GAME	1660-1740
INITIALIZE GRAPHIC CHARACTERS	1750-1870

VARIABLES

X, Y = POSITION OF INVADERS
X1 = DIRECTION OF INVADERS MOVEMENT
L1 = LIVES
SC = SCORE
MP = LASER BASE POSITION
MX, MY = YOUR MISSILE POSITION
IL = # OF INVADER LINES ALIVE

```
 10 REM * INVADERS *
100 GOSUB 1750
110 CLS : PAPER0
120 FORI=3TO26
130 :  PLOT0 , I , CHR$(9)
140 :  IFI<19THENPLOT1 , I , CHR$(5)
150 :  IFI>16ANDI<22THENPLOT1 , I , CHR$(3)
160 :  IFI>21THENPLOT1 , I , CHR$(1)
170 NEXTI
180 X=2 : Y=3 : X1=1
190 LI=3 : SC=0
200 MP=15 : MY=26 : IX=0 : IY=0 : IL=4
210 PRINTCHR$(6)CHR$(17)CHR$(20)
220 LL$="^ ^ ^ "
230 PLOT4 , 0 , "score 00          lives ^ ^ ^"
240 A1$="  AB  AB  AB  AB  AB  AB  "
250 SP$="                          "
260 A2$="  CD  CD  CD  CD  CD  CD  "
270 A5$=A1$ : A6$=A2$
280 A3$="  EF  EF  EF  EF  EF  EF  "
290 A4$="  GH  GH  GH  GH  GH  GH  "
300 A7$=A3$ : A8$=A4$
310 B1$=A3$ : B2$=A4$
320 B3$=A1$ : B4$=A2$
330 B5$=B1$ : B6$=B2$
340 B7$=B3$ : B8$=B4$
350 X=2 : Y=3
360 PLOT8 , 18 , "QQQ      QQQ      QQQ"
370 PLOT7 , 19 , "QQQQQ    QQQQQ    QQQQQ"
380 PLOT6 , 20 , "QQ   QQ  QQ   QQ  QQ   QQ"
390 PLOT6 , 21 , "QQ   QQ  QQ   QQ  QQ   QQ"
400 PLOTMP+2 , 25 , "I" : PLOTMP+1 , 26 , "JKL"
410 X=X+X1 : IFX>12THENX1=-X1
420 IFX<3THENX1=-X1 : Y=Y+1
430 IL=-1
440 IFA1$<>SP$THENIL=1
450 IFA3$<>SP$THENIL=2
460 IFA5$<>SP$THENIL=3
470 IFA7$<>SP$THENIL=4
480 IFIL=-1THENGOTO240
490 IF IL<1 THEN GOTO610
500 PLOTX , Y-1 , SP$
510 PLOTX , Y , A1$ : PLOTX , Y+1 , A2$
520 IF IL<2 THEN GOTO610
530 PLOTX , Y+2 , SP$
540 PLOTX , Y+3 , A3$ : PLOTX , Y+4 , A4$
550 IF IL<3 THEN GOTO610
560 PLOTX , Y+5 , SP$
570 PLOTX , Y+6 , A5$ : PLOTX , Y+7 , A6$
580 IF IL<4 THEN GOTO610
590 PLOTX , Y+9 , A7$ : PLOTX , Y+10 , A8$
600 IF Y+(3*IL)>24 THEN GOTO1660
610 K=PEEK(520)
```

(12#) (26#) (8#) (6#) (4#) (3#)

Where the number of blank spaces inside of quotation marks is critical or not easily determined we have noted the number of spaces thus: (12#).

```
 620 IFK<>172ANDK<>188ANDK<>132THENGOTO710
 630 MP=MP+(K=172)-(K=188)
 640 IFMP>34THENMP=34
 650 IFMP<2THENMP=2
 660 PLOTMP+1 , 25 , " I "
 670 PLOTMP ,  26 , " JKL "
 680 IFK<>132ORMY<26THENGOTO710
 690 SHOOT
 700 MX=MP+1 : MY=23
 710 IFMY>24THENGOTO760
 720 PLOTMX , MY , "   "
 730 MY=MY-1 : IFMY<4THENMY=26 : GOTO760
 740 IFSCRN(MX , MY)<>32ORSCRN(MX+1 ,
     MY)<>32THENGOSUB1270 : GOTO760
 750 PLOTMX , MY , "MN"
 760 IFIY=0THENGOTO830
 770 PLOTIX , IY , "   "
 780 IY=IY+1
 790 IF IY>26 THEN IY=0 : GOTO870
 800 IF SCRN(IX , IY)<>32ORSCRN(IX+1 , IY)<>32 THEN
     GOSUB1580 : GOTO870
 810 PLOT IX , IY , "OP"
 820 GOTO870
 830 IF RND(1)>.6 THEN GOTO870
 840 ZAP
 850 IY=Y+3*IL
 860 IX=X+2+4*INT(RND(1)*6)
 870 IF IL<1 THEN GOTO990
 880 PLOTX , Y-1 , SP$
 890 PLOTX , Y , B1$ : PLOTX , Y+1 , B2$
 900 IF IL<2 THEN GOTO990
 910 PLOTX , Y+2 , SP$
 920 PLOTX , Y+3 , B3$ : PLOTX , Y+4 , B4$
 930 IF IL<3 THEN GOTO990
 940 PLOTX , Y+5 , SP$
 950 PLOTX , Y+6 , B5$ : PLOTX , Y+7 , B6$
 960 IF IL<4 THEN GOTO990
 970 PLOTX , Y+8 , SP$
 980 PLOTX , Y+9 , B7$ : PLOTX , Y+10 , B8$
 990 K=PEEK(520)
1000 K=PEEK(520)
1010 IFK<>172ANDK<>188ANDK<>132THENGOTO1100
1020 MP=MP+(K=172)-(K=188)
1030 IFMP>34THENMP=34
1040 IFMP<2THENMP=2
1050 PLOTMP+1 , 25 , " I "
1060 PLOTMP , 26 , " JKL "
1070 IFK<>132ORMY<26THENGOTO1100
1080 SHOOT
1090 MX=MP+1 : MY=23
1100 IFMY>24THENGOTO1150
1110 PLOTMX , MY , "   "
1120 MY=MY-1 : IFMY<4THENMY=26 : GOTO1150
1130 IFSCRN(MX , MY)<>32ORSCRN(MX+1 ,
     MY)<>32THENGOSUB1270 : GOTO1150
1140 PLOTMX , MY , "MN"
```

```
1150 IFIY=0THENGOTO1220
1160 PLOTIX , IY , "  "
1170 IY=IY+1
1180 IF IY>26 THEN IY=0 : GOTO1260
1190 IF SCRN(IX , IY)<>32ORSCRN(IX+1 , IY)<>32 THEN
     GOSUB1580 : GOTO1260
1200 PLOT IX , IY , "OP"
1210 GOTO1260
1220 IF RND(1)>.7THEN GOTO1260
1230 ZAP
1240 IY=Y+3*IL
1250 IX=X+2+4*INT(RND(1)*6)
1260 GOTO410
1270 IFSCRN(MX , MY)=81ORSCRN(MX+1 , MY)=81THENPLOTMX
     , MY , "  " : MY=26 : RETURN
1280 SC=SC+10 : PLOT9 , 0 , STR$(SC)
1290 EXPLODE
1300 IFMY<Y+8 THENGOTO1370
1310 A8$=LEFT$(A8$ , MX-X-1)+"  "+RIGHT$(A8$ ,
     23-(MX-X))
1320 A7$=LEFT$(A7$ , MX-X-1)+"    "+RIGHT$(A7$ ,
     23-(MX-X))
1330 B8$=LEFT$(B8$ , MX-X-1)+"    "+RIGHT$(B8$ ,
     23-(MX-X))
1340 B7$=LEFT$(B7$ , MX-X-1)+"    "+RIGHT$(B7$ ,
     23-(MX-X))
1350 IFA8$=SP$THENPLOTMX-1 , MY-1 , "  " : PLOTMX-1
     , MY , "    "
1360 GOTO1570
1370 IFMY<Y+5 THENGOTO1440
1380 A6$=LEFT$(A6$ , MX-X-1)+"    "+RIGHT$(A6$ ,
     23-(MX-X))
1390 A5$=LEFT$(A5$ , MX-X-1)+"    "+RIGHT$(A5$ ,
     23-(MX-X))
1400 B6$=LEFT$(B6$ , MX-X-1)+"    "+RIGHT$(B6$ ,
     23-(MX-X))
1410 B5$=LEFT$(B5$ , MX-X-1)+"    "+RIGHT$(B5$ ,
     23-(MX-X))
1420 IFA6$=SP$THENPLOTMX-1 , MY-1 , "    " : PLOTMX-1
     , MY , "    "
1430 GOTO1570
1440 IFMY<Y+2 THENGOTO1510
1450 A4$=LEFT$(A4$ , MX-X-1)+"    "+RIGHT$(A4$ ,
     23-(MX-X))
1460 A3$=LEFT$(A3$ , MX-X-1)+"    "+RIGHT$(A3$ ,
     23-(MX-X))
1470 B4$=LEFT$(B4$ , MX-X-1)+"    "+RIGHT$(B4$ ,
     23-(MX-X))
1480 B3$=LEFT$(B3$ , MX-X-1)+"    "+RIGHT$(B3$ ,
     23-(MX-X))
1490 IFA4$=SP$THENPLOTMX-1 , MY-1 , "    " : PLOTMX-1
     , MY , "    "
1500 GOTO1570
1510 A2$=LEFT$(A2$ , MX-X-1)+"    "+RIGHT$(A2$ ,
     24-(MX-X))
```

```
1520 A2$=LEFT$(A2$ , MX-X-1)+"                "+RIGHT$(A2$ ,
     23-(MX-X))
1530 A1$=LEFT$(A1$ , MX-X-1)+"                "+RIGHT$(A1$ ,
     23-(MX-X))
1540 B2$=LEFT$(B2$ , MX-X-1)+"                "+RIGHT$(B2$ ,
     23-(MX-X))
1550 B1$=LEFT$(B1$ , MX-X-1)+"                "+RIGHT$(B1$ ,
     23-(MX-X))
1560 IFA2$=SP$THENPLOTMX-1 , MY-1 , "                " : PLOTMX-1
     , MY , "                "
1570 MY=26 : RETURN
1580 IFIY<25THENPLOTIX , IY , "  " : IY=0 : RETURN
1590 IX=0 : IY=0
1600 EXPLODE
1610 FORI=0TO7 : PAPERI : WAIT10 : NEXTI
1620 PAPER0
1630 LI=LI-1 : IFLI=0THENGOTO1660
1640 PLOT30 , 0 , LEFT$(LL$ , LI*2)+"  "
1650 RETURN
1660 PRINTCHR$(20)CHR$(17)CHR$(6)
1670 CLS : INK2
1680 PLOT10 , 10 , "BAD LUCK!!"
1690 PLOT3 , 11 , "The earth has been overrun"
1700 PLOT3 , 15 , "Do you want to try again ?"
1710 GET A$
1720 IF A$="Y"THENGOTO240
1730 IFA$<>"N"THENGOTO1710
1740 CLS : END
1750 FORI=47624TO47624+(17*8)-1
1760  :   READX : POKEI , X
1770 NEXTI
1780 DATA0 , 3 , 7 , 13 , 25 , 49 , 19 , 31 , 0 ,
     48 , 56 , 44 , 38 , 35 , 50 , 62
1790 DATA15 , 31 , 19 , 16 , 16 , 24 , 0 , 0 , 60 ,
     62 , 50 , 2 , 2 , 6 , 0 , 0
1800 DATA0 , 3 , 31 , 63 , 35 , 33 , 33 , 51 , 0 ,
     48 , 62 , 63 , 49 , 33 , 33 , 51
1810 DATA30 , 28 , 30 , 19 , 57 , 40 , 56 , 0 , 30 ,
     30 , 28 , 55 , 37 , 7 , 0 , 0
1820 DATA0 , 0 , 12 , 12 , 12 , 12 , 18 , 33 , 3 ,
     15 , 15 , 28 , 24 , 48 , 48 , 0
1830 DATA63 , 63 , 63 , 30 , 0 , 0 , 0 , 0 , 48 ,
     60 , 60 , 14 , 6 , 3 , 3 , 0
1840 DATA1 , 1 , 1 , 3 , 7 , 15 , 30 , 60 , 32 , 32 ,
     32 , 48 , 56 , 60 , 30 , 15
1850 DATA59 , 29 , 15 , 7 , 3 , 1 , 0 , 0 , 55 , 46 ,
     60 , 56 , 48 , 32 , 0 , 0
1860 DATA63 , 63 , 3 , 63 , 63 , 63 , 63 , 63
1870 RETURN
```

North Sea Copter

The object of this game is to pilot the helicopter (which is dangerously low on fuel) through a raging storm onto the oil rig platform.

This storm is so fierce and the rain so heavy that visibility is absolutely nil. You can see the beacon on top of the rig tower but the rig itself is totally obscured. When you fly into the path of the light's beam you will see it light up the copter.

The frequent lightning, accompanied by deafening thunder, briefly lights up the rig for you so that you can orientate yourself.

When the game starts it opens with a display of the crash routine where the helicopter breaks in two pieces which plummet into the sea. This is so that good pilots will still be able to see what happens to bad ones without sabotaging their craft.

The arrow keys move you left and right, the space bar gives you lift. Gravity pulls you down and all you have to do is regulate your drop by applying lift in the correct quantities remembering of course, that you have a very limited amount of fuel.

HOW THE PROGRAM WORKS

This game has been completely executed in the LORES mode using both the standard and alternate character sets simultaneously. The standard characters used have all been redefined using the Graphics Aid Program which you will find in this book. There was no particular reason for using both character sets at one time other than to show that it can be done and how it was done.

Executing the graphics in LORES is necessary to maintain speed of movement in the helicopter.

If you use more than one character to create an object on the screen and you also wish to move that object at a reasonable speed (and without being able to see the character blocks being redrawn one at a time) you will need to use LORES.

ORIC software only allows you to put one character at a time onto the screen (using CHAR). Therefore moving a block can only be done one character at a time and this totally destroys any illusion of animation or movement.

THE SEA

The sea surface is a row of the character 'S' whose shape has been redefined. The rest of the sea is the alternate 'Pound' character, which is a solid character block. From the surface to the bottom of the sea the far left screen attributes have been set to blue. (This explains the gap between the sea and the edge of the screen).

TOWER LIGHT

The beam of light that passes over the copter is achieved by setting the serial attribute at the height of the light (and on the far left) to 7 (i.e. white). It doesn't need to flash as the helicopter is already flashing and thus making the light flash too. The character which represents the light flashes on and off above the tower. This effect was a bonus that we got by not wanting to reset the attributes to the right of the tower as doing so would lose some room on the tower's right.

LIGHTNING

The lightning effect is simply a PAPER7 (white) followed immediately by a PAPER0 (Black) and is randomly generated.

THUNDER

The thunder makes use of the EXPLODE command and is syncronised with the lightning. After the explode command has been executed, a variable is decremented during each pass of the main program loop and the program ensures that all other sounds are 'skipped' until the variable reaches zero. This was necessary to sustain the thunder.

THE COPTER

The copter uses the redefined characters 'ACDE' and 'FGHI' ('B' is part of the tower). 'ACDE' is the copter with the main rotor blades and tail rotor. Switching back and forth between the two characters gives the effect of spinning rotors.

THE TOWER

The tower is constructed using characters from the alternate character set. Each horizontal layer of the tower is read from data into an array that is PLOTTED to screen during each display set-up.

STRUCTURE	LINE(s)
Initialization	100 - 170
Display set-up	200 - 240
Random lightning and thunder	320
Test for collision	325
Put copter (with rotor blades)	340
Turn on copter sound	420
Get keyboard character	430
Amend copter's velocity	440 - 460
Put alternate copter (without rotor blades)	480
Turn off copter sound	490
Calculate copter's new position	500 - 585
Test if on pad	600 - 630
On pad routine	700 - 790
Crash routine	900 - 990
Data	1000 - 2040

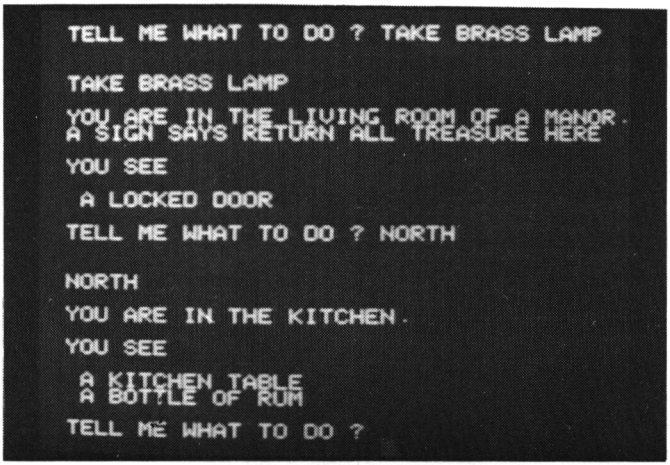
VARIABLES

PX = Copter's horizontal position
PY = Copter's vertical position
XV = Copter's horizontal velocity
YV = Copter's vertical velocity
MX = Copter's old horizontal position
MY = Copter's old vertical position
F = Fuel
PI = Tower's position
PAD = Pad's position

```
 50 REM **********************
 52 REM * NORTH SEA COPTER *
 54 REM **********************
 55 REM
 56 REM * LOW-RESOLUTION VERSION *
 57 REM
100 REM * INITIALIZATION *
110 CLS : POKE 618 , 10
112 FOR D=1 TO 10 : READ D$ : FOR I=0 TO 7
113 POKE 46080+ASC(D$)*8+I ,  PEEK
    (47104+ASC(D$)*8+I) : NEXT I , D
114 DATA b , z , q , \ , # , 7 , U , P , 9 , £
120 FOR C=1 TO 9 : READ C$(C) : V=46080+ASC(C$(C))*8
130 FOR I=V TO V+7 : READ D : POKE I , D : NEXT I , C
140 FOR P=0 TO 10 : READ P$(P) : NEXT P
144 CLS : PAPER1 : INK 7
145 PX=15 : PY=5 : GOTO 900
150 REM * START *
151 CLS : INK1 : PAPER0
153 P1=5
157 FOR P=0 TO 10 : PLOTP1 , P+9 ,
    CHR$(0)+P$(P)+CHR$(1) : NEXT
160 PX=30 : PY=5 : XV=.2 : YV=.2 : PAD=5 : MX=PX :
    MY=PY : F=20
170 L=15 : W=10 : R=L+W : DV=0
200 REM * SET-UP DISPLAY *
215 PLOT 0 , 8 , CHR$(7) : REM LIGHT
219 PLOT 0 , 20 , 6
220 PLOT 1 , 20 ,
    "sssssssssssssssssssssssssssssssssssss" : REM *
    SEA *
225 FOR K=21TO26 : PLOT 0 , K , 4
227 PLOT1 , K ,
    "ffffffffffffffffffffffffffffffffffffff" : NEXT
240 PLOT 5 , 25 , CHR$(3)+"GAS
    "+CHR$(1)+"ffffffffffffffffffffffff"
300 REM * MAIN LOOP *
310 F=F-.1 : IF F<0 THEN F=0
315 PLOT F+11 , 25 , CHR$(4)
319 TF=TF-1
320 IF RND(1)<.09THENPAPER7 : TF=5 : EXPLODE
323 PLOT P1+1 , 8 , "."
325 PLOTMX , MY , "    " : T=0 : FOR K=0 TO 3 :
    T=T+(SCRN(PX+K , PY)<>32) : NEXT
330 IF T<>0 THEN 600
340 PLOTPX , PY , "acde"
400 RE * GET A KEY *
410 IF F<1 THEN PLAY 3 , 5 , 6 , 1000 : GOTO 485
420 IF TF<1THEN PLAY 0 , 0 , 0 , 0 : PLAY 3 , 5 , 7 ,
    6
430 A$=KEY$ : IFA$=""THEN XD=0 ELSE A=ASC(A$)
440 IF A$=" "THEN DV=DV-YV : XD=0 : GOTO480
450 IF A9 THEN XD= XV
460 IF A=8 THEN XD=-XV
480 PLOT PX , PY , "fghi"  : PLOT P1+1 , 8 , " "
483 PAPER 0
```

```
485 IF TF>1 OR F<=0 THEN 500
490 PLAY 0 , 0 , 0 , 0
500 REM * MOVE COPTER *
510 IFDV<-YVTHENDV=-YV
520 DV=DV+YV/8
530 MY=PY : PY=PY+DV
540 MX=PX : PX=PX+XD
580 IF PX>33 THEN PX=33 : XV=0
585 IF PY<1 THEN PY=1 : DV=0
599 GOTO300
600 REM COPTER ON PAD ???
620 IF INT(PY)<16 OR INT(PY)>16 THEN 900
630 IFINT(PX)<INT(P1)+5 OR INT(PX)>INT(P1)+9 THEN 900
700 REM * ON PAD ROUTINE *
705 IF DV>.2 THEN 900
710 PAPER 6 : PLOT MX , MY , "fghi"
720 PLOT15 , 3 , "LANDED !"
721 FOR T=1 TO 5000 : NEXT
790 GOTO 150
800 REM * COLLISION TEST *
801 GOTO 300
900 REM COLLISION
903 IF SCRN(PX , PY)=ASC("S")THEN PLOT PX , PY ,
    "acde" : GOTO 977 : REM SPLASH
905 EXPLODE  : PAPER1 : WAIT 50 : PLAY 3 , 5 , 6 ,
    1000
910 X=PX
920 FOR Y=PY TO 25
930 X=X-1 : PX=PX+1
940 IF X<1 OR PX>33 THEN Y=25 : GOTO 970
945 PLOT X+1 , Y-1 , "  " : PLOT PX-1 , Y-1 , "  "
950 PLOT X , Y , "ac" : PLOT PX , Y , "de"
90 INK7 : PAPER1 : WAIT10 : INK1 : PAPER7 : NEXT Y
977 CLS : PAPER4 : EXPLODE  :WAIT 5
978 PLAY 0 , 0 , 0 , 0
979 PLAY 0 , 1 , 1 , 10000
988 WAIT 300
990 GOTO 150
1000 REM CHARACTER DATA
1010 DATA i , 0 , 9 , 2 , 62 , 4 , 9 , 0 , 0
1020 DATA h , 60 , 0 , 0 , 63 , 62 , 0 , 0 , 0
1030 DATA g , 63 , 16 , 56 , 23 , 27 , 23 , 62 , 60
1040 DATA f , 63 , 0 , 3 , 5 , 25 , 51 , 63 , 31
1045 REM
1050 DATA e , 0 , 0 , 0 , 62 , 0 , 0 , 0 , 0
1060 DATA d , 0 , 0 , 0 , 63 , 62 , 0 , 0 , 0
1070 DATA c , 0 , 16 , 56 , 31 , 31 , 31 , 62 , 60
1080 DATA a , 0 , 0 , 3 , 5 , 25 , 51 , 63 , 31
1085 REM
1090 DATA s , 12 , 4 , 15 , 31 , 63 , 63 , 63 , 63
2000 REM PLATFORM
2010 DATA b , z9 , zq , z9 , zq , z9 , zq , z9  \\\\\
2020 DATA b#7#7#7b
2030 DATA bPUPUPUb
2040 DATA b       b
```

Adventure


TELL ME WHAT TO DO ? TAKE BRASS LAMP
TAKE BRASS LAMP
YOU ARE IN THE LIVING ROOM OF A MANOR.
A SIGN SAYS RETURN ALL TREASURE HERE
YOU SEE
A LOCKED DOOR
TELL ME WHAT TO DO ? NORTH
NORTH
YOU ARE IN THE KITCHEN.
YOU SEE
A KITCHEN TABLE
A BOTTLE OF RUM
TELL ME WHAT TO DO ?

An adventure is a program that allows you to enter and explore strange new worlds, find treasures and battle fierce monsters. As an adventurer you must solve the riddles and problems of the world you will enter, or you will never leave the world alive.

This program allows you to have an adventure on your ORIC. The computer will describe the locations you will visit and the things you will see. Also it will prompt you to tell it what to do. The commands you may use consist of one or two words. You must always supply a verb (i.e. the action you wish to do) and often you will need a noun (i.e. the object you wish to perform the action on).

Some examples are :

'north' - move north and have a look around
'take axe' - if there is an axe where you are this will tell the computer to try and pick it up for you.

In a few cases you will be asked for more information after you have given the computer a command.

For example, if you say 'kill dragon' the computer will ask you what you wish to kill the dragon with. In this case just enter the name of the weapon you wish to use or nothing (just hit enter) if you wish to use your bare hands.

As well as giving you an adventure to play, this program also allows you to easily create your own worlds for yourself or a friend to adventure in.

HOW TO GO ON AN ADVENTURE

The computer will display a description of your location and any objects which exist there that you can see. You will then be asked what you want to do. Your command consists of one or two words. You must always supply a verb in your command, i.e., your action, 'NORTH', 'KILL DRAGON', 'SWIM". You may also be asked for more information in some cases. If you misspell your command or it is not understood you will be notified to try again.

Your commands are: NORTH, SOUTH, EAST, WEST, UP, DOWN, TAKE, DROP, KILL, INVENTORY, SCORE, CUT, SWIM, UNLOCK, DIG, EAT, WAKE, JUMP, DRINK.

These commands can all be abbreviated in some circumstances down to one letter. Where any ambiguity exists the first word found will be the command used. If an object has an adjective in it, e.g. MEAN TROLL, then you cannot say 'KILL TROLL', you must say 'KILL MEAN TROLL', or just 'KILL M' or 'K M'.

When playing an adventure it is usually a good idea to draw a map of the world you are exploring. Since many adventures contain mazes it is almost impossible to find your way around unless you have a good map showing you which way to go.

HOW THE PROGRAM WORKS

This program consists of two distinct parts. Firstly there are the descriptions, objects and actions that make up the game. Secondly there is the program needed to allow the player to interact with the data base. This consists of the initialization for the game, getting the user's input and performing the desired action.

THE DATA BASE

The fundamental parts of this adventure are the locations that make up the world, a map describing how these locations are connected, the objects that exist in this world and all the actions that the player may perform. The way that each of these is built up is now described in more detail.

Locations
For each location we need a description to give the player to tell him where he is and a list of the locations he can get to from where he is. As well as these it is often useful to have some sort of condition that can be applied upon entering a location. These can be anything from, making sure the player has a source of light before allowing him to enter, to performing some action when he enters (such as making monsters attack him if he is not carrying a particular object).

In this program each location has a unique number that the computer uses to access that location. The numbers start at one and increase in increments of one up to the last location. For example this adventure has 30 locations.

The locations exist in the program in a number of arrays. The array L$ contains the strings that describe the locations, i.e. the element L$(1) is the description for location 1. The 2 dimensional array L contains the

locations that the player will enter if he tries to move in one of the 6 directions N, S, E, W, U, D, i.e. L(1,1) is the location north of location 1 L(1,2) is south of location 1 etc. A value of 0 means that there is no location in that direction. The array L contains another element for each location (e.g. L(1,7) that defines the CONDITION that must be applied for the player to ENTER THIS location. These conditions consist of short pieces of program that are located at lines 4000-4208 for this adventure.

Objects

The information we need for each object consists of its description, where it is, what type of object it is (i.e. a treasure or a monster etc), how strong it is and to allow greater flexibility a condition to be performed if the player tries to pick up this object. These conditions may be one that always fails to stop people from taking things like tables, or one to kill the player if he tries to take a forbidden object.

The computer knows about each object by its number just like locations.

The data for the objects is contained in two arrays OD$ and OD. OD$ contains the object description as for the locations while OD is a two dimensional array that contains the following information:

First element	-	location of the object (0 if carried by the player, -1 if it is dead.or destroyed)
Second element	-	0 for an ordinary object <0 for a monster >0 for a treasure and this is the score for the treasure
Third element	-	strength for a weapon this is added to the player's strength in a fight and for a monster it is the strength of the monster
Fourth element	-	the condition to be done if the player takes the object

Lastly we need a list of actions that the player can perform. These are stored in the array V$ so again each verb will have a number associated with it and in this case the number is used to find the subroutine that will perform the action for the player.

Clearly all these arrays need to be set up and this is done at the start of the program by reading in the strings and values contained in the large block of data statements at the end of the program. To make things more easily changed the number of actions, objects, and locations are stored in the data statements and are read in first. These data statements will be explained in more detail later.

STRUCTURE	LINE(S)
Initial message and tune subroutine	1 - 4
Call subroutine to set up data arrays	10
This is the control loop for whole program. It consists of : 1) print location description and any visible objects present; 2) depending on location set new paper colour; 3) get players input command and perform the action. If the action failed goto 3 else goto 1.	20 - 210
This subroutine inputs the players command. The first two words that the player types are put into variables A$ and N$. (A word is any sequence of characters ending with a space). The verb array is then searched for A$ and if it is found the subroutine returns with the verb number to the main program. Otherwise the player is prompted to try again.	1000 - 1120
Subroutine that searches object description array OD$ for the word N$. (Actually searches for W$ so a call can be made to 1305 if W$ has been set to the required word. If the word is found it returns the description from the array, the object number and the flag 'F' indicating that the search was successful, otherwise the flag is set to indicate failure.	1300 - 1340
Consists of the subroutines to process all the actions that the player may perform. The appropriate subroutine is called by the GOSUB function in line 60 which uses two variables and a constant to calculate the required subroutine.	1500 - 1862
A jump is made here if an action fails, and the message 'you can't' is to be printed.	1999

Prints all the objects that are in location L, (IF L=0 then prints all objects carried by the player). If no objects found then print 'nothing'. 2000 - 2050

This subroutine checks for the correct conditions for entry or action using variable C. For this adventure C is in the ranges of 0-8 or 50 or 60.

0 always succeeds
1 always fails

2-8 are used for entering various locations and taking certain objects. 50 and 60 are subroutines to be performed and not just simple tests. 4000 - 4215

Verb Data :
The first number specifies the number of actions the player may perform. This number will also allow you to use as many data lines as you wish, (as many are quite long) to enter your data, the rest is the words that define the actions. 2000 - 7002

Object Data :
The first number is the number of objects in the game. Say 'n' there are then 'n' numbers that are the starting locations for each object. The next 'n' strings are the object descriptions. Lastly, there are '3n' numbers, where each triple contains the type, strength and condition for an object. 8000 - 8090

Location Data :
The first number is the number of locations in the Adventure. There is then one line for each location containing the object description, the locations that are connected to this one, and the condition for this location. 8100 - 8133

Reads all the data into the arrays 9000 - 9120

These lines are jumped to after a successful Adventure (the player finds all the treasure). 9980 - 9988

This is the ENDGAME (either after the player has been killed or if he finished the game). It prints the score and asks for another game. 9990 - 9994

```
  1 PRINT CHR$(12) : PRINT : PRINT CHR$(4) : PAPER0
    : INK7
  2 PRINT CHR$(27);" GOING ON AN ADVENTURE....";
    CHR$(4)
  4 GOSUB 10000
 10 PC=1480 : L=1 : GOSUB 9000 : CLS
 20 PRINT"YOU ARE ";L$(L) : Q$="YOU SEE" :
    PRINT : GOSUB 2000
 40 GOSUB 1000 : PRINT
 50 PRINT V$;" ";
 60 GOSUB PC+VN*20
 65 PRINT
 70 IF L>2 AND L<11 THEN PAPER3 : INK0 : GOTO 200
 80 IF L=25 THEN PAPER6 : PING : GOTO 200
 90 IF L=14 THEN PAPER0 : INK1 : GOTO 200
100 IF L=17 THEN PAPER5 : INK0 : GOTO 200
110 IF L=12 THEN PAPER2 : INK0 : GOTO 200
130 IF L=20 THEN EXPLODE : FOR J=1 TO 20 :
    PAPER RND(1)*7 : WAIT5 : NEXT : GOTO200
140 PAPER 4 : INK 7
200 IF F=1 THEN 20
210 PRINT : GOTO 40
1000 INPUT"TELL ME WHAT TO DO ";S$ : PRINT :
     IF LEN(S$)=0 THEN 1000
1010 A$="" : N$=""
1020 X=0 : FOR I=1 TO LEN(S$)
1030 I$=MID$(S$ , I , 1)
1050 IF I$<>" " THEN 1065
1055 IF X<>1 THEN 1065
1060 I=LEN(S$) : GOTO 1080
1065 IF I$=" " THEN X=1 : GOTO 1080
1070 IF X=0 THEN A$=A$+I$ : GOTO 1080
1075 N$=N$+I$
1080 NEXT I : VN=0
1090 FOR I=1 TO VC : X=LEN(V$(I)) : IF X>LEN(A$)
     THEN X=LEN(A$)
1100 IF LEFT$(A$ , X)=LEFT$(V$(I) , X)THEN VN=I :
     V$=V$(I) : I=VC
1110 NEXT I : IF VN=0 THEN PRINT"I DONT UNDERSTAND
     ";A$ : GOTO 1000
1120 RETURN
1300 W$=N$ : GOSUB 1305 : IF F=0 THEN RETURN ELSE
     PRINT Y$ : RETURN
1305 VN=0 : IF LEN(W$)=0 THEN 1335
1310 FOR I=1 TO OC : X=LEN(OD$(I)) : IF X>LEN(W$)
     THEN X=LEN(W$)
1320 IF LEFT$(W$ , X)=LEFT$(OD$(I) , X)THEN VN=I :
     Y$=OD$(I) : I=OC
1330 NEXT I
1335 IF VN=0 THEN PRINT"I DONT UNDERSTAND ";W$ :
     F=0 : RETURN
1340 F=1 : RETURN
1500 D=1 : GOTO 1601
```

```
1520 D=2 : GOTO 1601
1540 D=3 : GOTO 1601
1560 D=4 : GOTO 1601
1580 D=5 : GOTO 1601
1600 D=6
1601 PRINT : IF L(L , D)=0 THEN PRINT"YOU CANT GO
     THAT WAY!" : F=0 : RETURN
1603 C=L(L(L , D) , 7) : GOSUB 4000
1605 IF F=0 THEN 1615
1610 IF OD(1 , 1)=0 THEN L=L(L , D) : F=1 : RETURN
1615 PRINT"SOMETHING STOPS YOU." : F=0 : RETURN
1620 GOSUB 1300 : IF F=0 THEN RETURN
1621 IF OD(VN , 1)<>L THEN PRINT "IT'S NOT HERE." :
     F=0 : RETURN
1622 IF OD(VN , 2)<0 THEN 1999
1625 C=OD(VN , 4) : GOSUB 4000 : IF F=0 THEN 1999
1630 IF CX<>0 THEN CX=CX-1 : OD(VN , 1)=0 : RETURN
1631 PRINT"YOU ARE CARRYING TOO MUCH." F=0 : RETURN
1640 GOSUB 1300 : IF F=0 THEN RETURN
1641 IF OD(VN , 1)<>0 THEN PRINT"YOU DONT HAVE IT."
     : F=0 : RETURN
1642 OD(VN , 1)=L
1643 IFL=1ANDOD(VN , 2)<>0THENSC=SC+OD(VN , 2) :
     OD(VN , 1)=-1 : PRINTY$;" VANISHES"
1644 CX=CX+1 : IF SC=220 THEN 9980
1645 RETURN
1660 GOSUB 1300 : IF F=0 THEN RETURN
1661 T$=Y$ : MN=VN : IF OD(VN , 2)=>0 THEN 1999 ELSE
     MS=OD(VN , 3)
1662 S=0 : PRINT : INPUT"KILL WITH WHAT";W$ :
     IF W$="" THEN 1670
1663 GOSUB 1305 : IF F=0 THEN 1662
1664 IF OD(VN , 1)<>0 THEN PRINT"YOU DONT HAVE IT" :
     GOTO 1662
1665 IF OD(VN , 3)=0 THEN PRINT"IT'S NOT A WEAPON" :
     GOTO 1662
1666 S=OD(VN , 3)
1667 GOSUB 10000 : OD(MN , 1) =-1 : F=1 : RETURN
1670 PRINT : S=S+ST : IF MS>S+RND1)*20 THEN 1679
1671 IF S>MS+RND(1)*15 THEN 1675
1672 PRINT"THE "  ;T$ : PRINT"FIGHTS BACK.YOU FEEL
     WEAKER."
1673 ST=ST-RND(1)*5 : F=1 : RETURN
1675 PRINT"YOU HAVE KILLED THE ";T$
1676 PRINT"THE BODY VANISHES IN A CLOUD OF SMOKE."
1677 OD(MN , 1)=-1 : F=1 : RETURN
1679 PRINT"THE ";T$;" HAS KILLED YOU" : GOTO 9990
1680 L1=L : L=0 : PRINT
1685 Q$="YOU ARE CARRYING"
1690 GOSUB 2000 : L=L1 : F=1 : PRINT : RETURN
1700 PRINT SC : F=1 : RETURN
1720 GOSUB 1300 : IF F=0 THEN RETURN
1721 IF VN<>20 OR OD(2 , 1)<>0 THEN 1999
1722 OD(14 , 1)=OD(VN , 1) : OD(VN , 1)=-1 :
     PRINT"AN IVORY KEY FALLS." : RETURN
```

```
1723 PRINT"AN IVORY KEY FALLS FROM THE CIELING" :
     RETURN
1740 PRINT : IF L<>11 AND L<>29 THEN1999
1741 IFL=11ANDOD(3 , 1)<>0THENPRINT"YOU SINK TO THE
     BOTTOM AND DROWN" : GOTO9990
1742 L=40-L : F=1 : RETURN
1760 GOSUB 1300 : IF F=0 THEN RETURN
1761 IF L=1 AND OD(4 , 1)=0 THEN OD(22 , 1)=-1 :
     OD(23 , 1)=L : LU=1 : RETURN
1762 IF L=13 AND OD(14 , 1)=0 THEN OD(24 , 1)=-1 :
     OD(25 , 1)=L : GU=1 : RETURN
1763 GOTO 1999
1780 IF OD(16 , 1)<>0 OR L<3 OR L>10 THEN 1999
1781 PRINT"YOU FIND ";
1782 IF L=6 AND OD(4 , 1)=-1 THEN OD(4 , 1)=L :
     PRINT"SOMETHING" : F=1 : RETURN
1783 PRINT"NOTHING" : F=1 : RETURN
1800 GOSUB 1300 : IF F=0 THEN RETURN
1801 IF OD(VN , 2)<0 THEN 1999
1802 C=OD(VN , 4) : GOSUB 4000 : IF F=0 THEN 1999
1803 PRINT"THAT HITS THE SPOT." : OD(VN , 1)=-1 :
     RETURN
1820 GOSUB 1300 : IF F=0 THEN RETURN
1821 IF L<>24 OR VN<>7 THEN 1830
1822 PRINT"A BRIDGE APPEARS ACROSS THE CHASM."
1825 OD(15 , 1)=L : BE=1 : F=1 : RETURN
1830 PRINT"NOTHING HAPPENS." : F=1 : RETURN
1840 IF L=24 OR L=26 THEN PRINT"THE FALL HAS BROKEN
     YOUR NECK." : GOTO 9990
1841 GOTO 1830
1860 GOSUB 1300 : IF F=0 THEN RETURN
1861 IF VN<>18 THEN 1999
1862 PRINT"THAT WAS DELICIOUS." : OD(19 , 1)=OD(VN ,
     1) : OD(VN , 1)=-1 : RETURN
1999 PRINT"YOU CAN'T." : F=0 : RETURN
2000 X=0 : PRINT Q$  : PRINT
2010 FOR I=1 TO OC : IF L<>OD(I , 1) THEN 2030
2020 X=1 : PRINT" A";
2022 Z$=LEFT$(OD$(I),1)
2023 IF Z$="A" OR Z$="E" OR Z$="I" OR Z$="O" OR
     Z$="U" THEN PRINT"N";
2024 PRINT" ";OD$(I)
2030 NEXT I
2040 IF X=0 THEN PRINT" NOTHING."
2050 PRINT : RETURN
4000 IF C=>50 THEN ON C/10-4 GOTO 4100 , 4110
4010 ON C GOTO 4201 , 4202 , 4204 , 4206 , 4208 ,
     4210 , 4212 , 4214
4020 F=1 : RETURN
4100 IF OD(18 , 1)<>0 THEN F=1 : RETURN
4101 OD(18 , 1)=-1 : OD(12 , 1)=-1 : PRINT
4102 PRINT"AS YOU ENTER A PIRATE STEALS YOUR RUM , "
4103 PRINT"AND RUNS OFF LAUGHING" : F=1 : RETURN
4110 PRINT"THE DEVIL KILLS YOU." : GOTO 9990
4201 F=0 : RETURN
4202 IF LU<>0 THEN F=1 ELSE  F=0 : RETURN
```

```
4203 RETURN
4204 IF GU<>0 THEN F=1 ELSE F=0 : RETURN
4205 RETURN
4206 IF BE<>0 THEN F=1 ELSE F=0 : RETURN
4207 RETURN
4208 IF OD(3 , 1)=0 THEN F=1 ELSE F=0 : RETURN
4209 RETURN
4210 IF OD(11 , 1)=-1THEN F=1 ELSE F=0 : RETURN
4211 RETURN
4212 IF OD(12 , 1)=-1THEN F=1 ELSE F=0 : RETURN
4213 RETURN
4214 IF OD(10 , 1)=-1 THEN F=1 ELSE F=0 : RETURN
4215 RETURN
7000 DATA 19 , NORTH , SOUTH , EAST , WEST , UP ,
     DOWN
7001 DATA TAKE , DROP , KILL , INVENTORY , SCORE ,
     CUT , SWIM
7002 DATA UNLOCK , DIG , EAT , WAKE , JUMP , DRINK
8000 DATA 25 , 1 , 12 , 27 , -1 , 14 , 17 , 21 ,
     25 , 29
8002 DATA 12 , 14 , 17 , 20 , -1 , -1 ,
     11 , 2 , 2 , -1
8004 DATA 19 , 11 , 1 , -1 , 13 , -1
8010 DATA BRASS LAMP , SWORD , SNORKEL
8012 DATA LARGE KEY , PERSIAN RUG , GOLD COIN
8020 DATA SILVER WAND , RUBY , DIAMOND
8022 DATA MEAN TROLL , GREEN DRAGON , PIRATE
8030 DATA DEVIL , IVORY KEY , CRYSTAL BRIDGE
8032 DATA SHOVEL , KITCHEN TABLE , BOTTLE OF RUM ,
     EMPTY BOTTLE
8040 DATA ROPE TIED BETWEEN FLOOR AND CIELING ,
     SMALL LAKE , LOCKED DOOR
8050 DATA DOOR , PADLOCKED GATE , GATE
8060 DATA 0 , 0 , 0 , 0 , 20 , 8 , 0 , 0 , 0 , 0 ,
     0 , 0 , 10 , 0 , 6 , 50 , 0 , 7 , 20 , 0 , 0
8070 DATA 30 , 0 , 0 , 100 , 0 , 0 , -1 , 87 , 0 ,
     -1 , 110 , 0 , -1 , 200 , 0 , -1 , 200 , 0 ,
     10 , 0 , 0
8080 DATA 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 ,
     0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 , 1
8090 DATA 0 , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 1 , 0 ,
     0 , 1
8100 DATA 30
8101 DATA IN THE LIVING ROOM OF A MANOR.A SIGN SAYS
     RETURN ALL TREASURE HERE
8102 DATA 2 , 0 , 0 , 0 , 0 , 12 , 0
8103 DATA IN THE KITCHEN. , 0 , 1 , 3 , 0 , 0 ,
     0 , 0
8104 DATA IN AN ENDLESS DESERT. , 0 , 4 , 5 , 2 ,
     0 , 0 , 0
8105 DATA IN AN ENDLESS DESERT. , 3 , 7 , 6 , 0 ,
     0 , 0 , 0
8106 DATA IN AN ENDLESS DESERT. , 0 , 6 , 7 , 3 ,
     0 , 0 , 0
8107 DATA IN AN ENDLESS DESERT. , 5 , 8 , 0 , 4 ,
     0 , 0 , 0
```

```
8108 DATA IN AN ENDLESS DESERT. , 0 , 0 , 0 , 4 ,
     0 , 0 , 0
8109 DATA IN AN ENDLESS DESERT. , 6 , 9 , 0 , 0 ,
     0 , 0 , 0
8110 DATA IN AN ENDLESS DESERT. , 8 , 0 , 10 , 0 ,
     0 , 0 , 0
8111 DATA IN AN ENDLESS DESERT. , 11 , 0 , 0 , 9 ,
     0 , 0 , 0
8112 DATA AT AN OASIS. , 0 , 10 , 0 , 0 , 0 , 0 , 0
8113 DATA IN A CELLAR. , 0 , 16 , 13 , 0 , 1 , 0 , 2
8114 DATA AT THE GATES OF HELL. , 0 , 0 , 0 , 12 ,
     0 , 20 , 0
8115 DATA IN A BLACKENED CAVERN. , 0 , 15 , 0 , 0 ,
     0 , 0 , 0
8116 DATA IN A PASSAGE. , 14 , 18 , 0 , 16 , 0 , 0 ,
     0
8117 DATA IN A PASSAGE. , 120 , 15 , 0 , 0 , 17 , 0
8118 DATA IN THE PIRATES LAIR. , 0 , 0 , 0 , 0 ,
     16 , 0 , 50
8119 DATA IN A PASSAGE. , 15 , 19 , 0 , 0 , 0 ,
     0 , 0
8120 DATA IN A PASSAGE. , 18 , 0 , 0 , 0 , 0 , 0 , 0
8121 DATA IN HELL.A DEVIL SAYS'FIND THE RIGHT
     DIRECTION AND LIVE ELSE DIE.'
8122 DATA 30 , 30 , 30 , 21 , 30 , 30 , 3
8123 DATA IN A PASSAGE. , 0 , 24 , 22 , 20 , 0 ,
     0 , 0
8124 DATA IN A DEAD END. , 0 , 0 , 0 , 21 , 0 ,
     0 , 0
8125 DATA DEAD IN HELL. , 0 , 0 , 0 , 0 , 0 , 0 , 0
8126 DATA AT THE BRINK OF A DEEP CHASM. , 21 , 25 ,
     0 , 0 , 0 , 0 , 0
8127 DATA IN A BEAUTIFUL JEWELLED HALL. , 24 , 0 ,
     27 , 0 , 0 , 0 , 4
8128 DATA AT THE BRINK OF A DEEP PIT.THERE ARE
     TRACES OF FIRE AND BRIMSTONE.
8129 DATA 26 , 0 , 0 , 25 , 0 , 28 , 0
8130 DATA IN A PASSAGE. , 26 , 0 , 0 , 25 , 0 , 28 ,
     0
8131 DATA IN A PASSAGE. , 0 , 0 , 0 , 0 , 27 ,
     18 , 0
8132 DATA SWIMMING IN A SMALL LAKE. , 0 , 0 , 0 ,
     0 , 0 , 0 , 5
8133 DATA  , 0 , 0 , 0 , 0 , 0 , 0 , 60
9000 RESTORE : READ VC
9010 DIM V$(VC)
9020 FOR I=1 TO VC : READ V$(I) : NEXT I
9030 READ OC : DIM OD(OC , 4) : DIM OD$(OC)
9040 FOR I=1 TO OC : READ OD(I , 1) : NEXT I
9050 FOR I=1 TO OC : READ OD$(I) : NEXT I
9060 FOR I=1 TO OC : FOR J=2 TO 4
9070 READ OD(I , J) : NEXT J : NEXT I
9080 READ LC : DIM L$(LC) , L(LC , 7)
9090 FOR I=1 TO LC : READ L$(I)
9100 FOR J=1 TO 7 : READL(I , J) : NEXT J : NEXT I
9110 SC=0 : BE=0 : LU=0 : GU=0 : CX=6 : ST=100
```

```
 9120 RETURN
 9980 PRINT : PRINT : PRINT
 9985 PRINT"CONGRATULATIONS , THIS ADVENTURE IS OVER
      AND YOU HAVE RETURNED."
 9988 PRINT"SAFELY WITH ALL THE TREASURES."
 9990 PRINT : PRINT"SCORE ";SC : PRINT"DO YOU WISH TO
      PLAY AGAIN ?"
 9991 GET M$ : IF M$="" THEN 9991
 9992 IF M$="Y" THEN RUN
 9993 PRINT"BYE."
 9994 END
10000 FOR J=12 TO 1 STEP -1
10010 FOR K=1 TO 6
10020 PLAY 1 , 0 , 3 , J*3000
10030 MUSIC 1 , K , RND(1)*12+1 , 10
10040 NEXT K
10050 NEXT J
10060 PLAY 0 , 0 , 0 , 0
10070 RETURN
```

HOW TO CREATE YOUR OWN ADVENTURE

Firstly you need a world to explore. This involves building into the data base the descriptions of all the locations to be visited along with the connections between them. You will also need to decide what, if any, conditions will be applied to entering the location.

As an example let us build an adventure with two locations. These will be:

A HALL STRETCHING AS FAR AS THE EYE CAN SEE
and
A SMALL CAVERN.

The hall is north of the cavern and going south from the hall ends up in the cavern. No other directions are allowed. As a condition there will be a door between the two locations that will be locked. The player can only pass between the two if the door has been unlocked. So we have :

```
8100 DATA 2 (the number of locations)
8101 DATA IN A HALL STRETCHING AS FAR AS THE EYE CAN SEE.,
     0,2,0,0,0,0,0
8102 DATA IN A SMALL CAVERN.,1,0,0,0,0,0,1
```

Note doors are only one way so that entering the hall has no condition while entering the cavern requires that the door be unlocked.

Since the hall is the first location this will be where the player starts and where treasures must be put.

Now we need some objects. We already need a locked door and an unlocked door (at the start the locked door will exist between the locations but when the door is unlocked we will put the locked door in location -1 to get rid of it and replace it with the unlocked one). Obviously we also need a key to unlock the door. For a treasure let us have an emerald in the cavern. As well there will be a sword in the hall and a snake guarding the emerald.

OBJECT	TYPE	STRENGTH	CONDITION	LOCATION
Locked door	0	0	fail	1
Unlocked door	0	0	fail	-1
Key	0	0	0	1
Sword	0	10	0	1
Snake	-1	100	fail	2
Emerald	30	0	2	2

This means that the player cannot take either the door or the snake, and the emerald can only be taken if condition 2 is met. We will write condition 2 as a test for the snake being alive.

The sword has a strength of 10 and the snake 100 so if the player starts with a strength of 100 the fight between them should be fairly matched with a slight advantage to the player.

The emerald has a score of 30, the snake is a monster and the rest are just objects.

In the program this would be :

```
8000 DATA 6,1,-1,1,1,1,2,2
8001 DATA LOCKED DOOR,UNLOCKED DOOR,KEY,SWORD,SNAKE
8002 DATA SNAKE,EMERALD
8003 DATA 0,0,1,0,0,1,0,0,0,0,10,0
8004 DATA -1,100,1,30,0,2
```

Now we need the actions to perform. The movements and their subroutines as well as 'INVENTORY', 'TAKE', 'DROP' and 'KILL' are fairly standard and will probably be used in all your adventures so look at the program for these. The other action we will have will be 'UNLOCK' for the door, so the verb list will be :

```
7000 DATA 11, NORTH,SOUTH,EAST,WEST,UP,DOWN
7001 DATA TAKE,DROP,KILL,INVENTORY,UNLOCK
```

As well line 60 of the program will need to be changed since this calls the subroutine to do the players action. You must decide where each subroutine will go and then fill in the addresses. (The first address is for the first verb, the second address is for the second verb etc).

The only routine we need to write is for unlock. This needs to check if the player is in the hall and is carrying the key and the door is still locked. We now need a variable to indicate if the door is locked or not say 'LU' (remember this must be initialized at the start or the door may be unlocked when we thought it was locked).

Unlocking, then will consist of :

```
1700 IF LU=1 OR L()1 OR OD(3,1)()0 THEN 1999
1701 OD(1,1)=-1: OD(2,1)=1: LU=1
1702 F=1: RETURN
```

A note about the variable 'F', it is used to indicate success or failure in the following manner :

```
F = 0   -   failure
F = 1   -   success
```

We still have 2 conditions and they are :

```
1   is the door unlocked
and
2   is the snake dead
```

The subroutine to test the right condition will be :

```
4000 ON C GOTO 4101,4102
4001 F=1: RETURN
```

And the conditions themselves :

```
4101 F = -(LU () 0): RETURN
4102 F = -(OD(5,1) = -1): RETURN
```

These may need some explanation. Take the second, what it means is IF OD(5,1) = -1 THEN F=1
else IF OD(5,1) () -1 THEN F=0

It works because the statement OD(5,1)()-1 will be equal to 0 if IF OD(5,1)=-1 and it will equal -1 otherwise.

So now we have an adventure even though it is pretty small. Good luck writing your own.

HINTS FOR PLAYING THE ADVENTURE PROGRAM

1) Lost in the desert?
Do not despair the desert is not really endless and if you strategically drop objects in the desert you should be able to make a map of the desert. You will need this map to find the oasis.

2) Can't get through the living room door?
You will need the key that is buried somewhere in the desert. To dig up the key you will of course need a shovel which is also outside the house - somewhere.

3) Stuck in HELL?
Do what the devil says.

4) Can't get past the pirate?
Try the rum.

5) Can't kill the dragon?
Keep trying.

6) Can't find the key to the gate?
Try the rope.

7) Can't cross the chasm?
Maybe a magic wand would be of assistance.

8) Can't find the last treasure?
Go swimming.

Utility Programs

The following section contains the Utility programs for the ORIC, as well as a detailed explanation of ORIC's Basic. Amongst other programs are Renumber and Block Delete, which were not written in Basic, but 6502 Machine Language.

Renumber and Block Delete were both written in 6502 Machine Language because:

Firstly, they both involve shifting the Basic program around in memory. ORIC's Basic cannot cope with executing a program that insists on shifting itself around! Because Machine Language is independent of the Basic System, this does not matter.

Secondly, Machine Language offers a huge speed advantage over Basic.

There are a few steps involved in keying in Renumber and Block Delete:

1. Power up ORIC

2. Key HIMEM #99FF

3. Load MACHINE CODE MONITOR & RUN it.

4. Enter a starting address of $9A00 for Renumber or $9E00 for Block Delete.

5. Key in code as shown on the relevant listing, as per the instructions for Machine Code Monitor.

6. Get out of MACHINE CODE MONITOR.

7. To save the new program on tape key
   a) For Renumber
      CSAVE "Renumber",A#9A00,E#9D40
   b) For Block Delete
      CSAVE "Block Del",A#9E00,E#9ED5

Oric1's Basic

The Basic Program:

The ORIC-1's Basic is structured in much the same manner as most other Microsoft "look-alike" Basics.

In page zero (hex $0000 - $00FF), live two 16 bit pointers:

1) A pointer to the Start of the Basic program (SOPP).

2) A pointer to the End of the Basic program (EOPP).

These pointers reside at locations $9A and $9C respectively, and both take up 2 bytes each because they are 16 bit pointers.

As it happens, when ORIC initially loads a Basic program, either from tape or from the keyboard, it begins storing it from $0501 on. Key in the following:

```
PRINT HEX$(DEEK(#9A))
```

You should have #501 on the screen if all is normal.

The complete Basic program comprises of individual records, which are structured as below:

LA	L#	Tokenised Basic Statements	EOL marker

LA - Is a Link Address pointing to the next record in the program. (Try and visualise the program as a chain, where each Link Address is actually a Link.) As is convention for 6502 systems, the low byte of the address comes first.

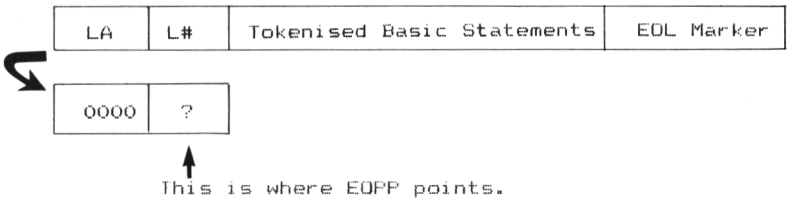
L# - Is the line number for that record. It will be contained as a Hex number and, as above, the low byte will come first.

Tokens - In a Casino you may use a single chip (not the legged variety!) to stand for several Pounds. ORIC's Basic uses "Tokens" in the same fashion to stand for several characters. It does this by substituting a number for a

string of characters, provided they form a word kept in ORIC's vocabulary (usually known as a Reserved Word). This number is not confused with other character bytes in the program because it has its high bit set (i.e. is >127 or $7F). For instance, the word "GOSUB" will be represented by single $9B. Strings of characters not forming a "Reserved Word" in ORIC's vocabulary will just be stored as normal ASCII values. A list of ORIC's Token Numbers and their respective Reserved Words can be obtained by running "TOKEN TABLE".

EOL - Is the End Of Line marker. As in most systems, The ORIC-1 uses a $00 for this purpose.

The End of the program is defined as being the byte following a Link Address of zero. It is important to remember that EOPP points to the byte after the zero Link Address.



Numeric & String Variable Storage:

ORIC picks up new variables and assigns storage for them as it encounters them during execution of your Basic program. When ORIC comes across a variable in your program it checks to see whether it has already been allocated storage, and if it hasn't, then ORIC does so. When first defined, the variable will have a null value (0 for numerics, "" for strings). Where ORIC allocates storage for a variable is dependant on the variable's "Type".

ORIC supports 3 different Types of variables:

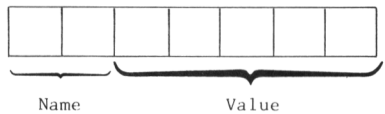
1) Floating Point

2) Floating Point Array

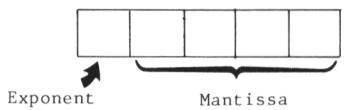
3) String

Normal Floating Point and String variables are allocated storage immediately following the Basic Program. Floating Point and String Arrays follow straight after. This means that if a new variable is defined after an Array, then that Array must be moved to make room for the new Floating Point or String variable to fit below it. This can take a long time if it involves a big array and several new variables.

Floating Point variables have 7 bytes allocated to them, shown as below:



The first 2 bytes contain the characters which form the variable's name. This is why recognizable variable names are limited to 2 characters. The remaining 5 bytes are used to contain the variable's value in Floating Point form.



Floating Point representation works in the following manner:

Take the number 1234.5 for example. We can also represent this as .12345 E4. In this case the Exponent would be 4 and the Mantissa .12345, using Base 10 (Decimal). ORIC, of course, uses Base 2 (Binary). The Binary representation of 1234.5 is:

010011010010.1 Binary

4D2.8 Hex

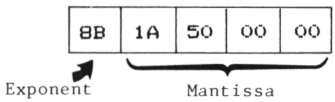
or

$0.100110100101 * 2^{B}$ Binary

$0.1A5 * 2^{B}$ Hex

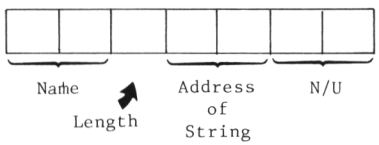
In this case, the Exponent is $B, and the Mantissa is $0.1A5. As it happens, ORIC adds $80 to the exponent. It does this so that if the size of a number is < 1, the Exponent will be < $80

(positive), whilst larger sizes will have an Exponent > $80 (negative). ORIC will also set the Most Significant Bit (MSB) of the Mantissa to indicate that the variable is negative, if that is the case. Therefore, our final Floating Point representation of 1234.5 is:



The same Floating Point form is used to represent the values of the individual elements of a Floating Point Array.

String variables also have 7 bytes of storage allocated to them as shown below:



As mentioned above, String and Floating Point variables mingle together between the Basic program and the Arrays. ORIC distinguishes String variables from Floating Point types by checking the MSB of the second character of the variable's name. If it is set, then the variable is a String. i.e. ZB$ will be saved as $5A, $C2, whereas A1 will be stored as $41, $31.

The 3rd byte of the variable gives the length of the string that the variable holds. The 4th & 5th bytes give the Address of the string. The 6th & 7th bytes are unused. Consider the following:

```
10 A$="hi there"
```

Once ORIC executes this statement, the pointer in the variable's storage will point to where the string actually appears in the above statement. In this way, unecessary copies of strings are avoided. If a String variable contains a value not given to it by a simple equate as shown above, then the actual string will be kept in an area of memory below HIMEM. (The value of HIMEM is kept in $A6 to $A7)

This area of memory starts at HIMEM, and works its way down as new strings are defined, and as old ones are discarded. You see, when old strings are discarded, they are not automatically "cleaned up" from this area, and so they accumulate until eventually, they fill up all of the available memory between

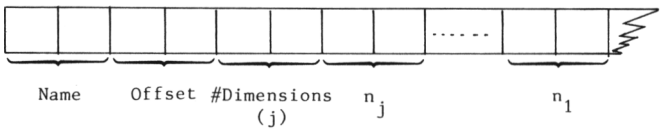
HIMEM and the last of the Arrays. Only when things reach this drastic stage does ORIC perform what is known as "Garbage Collection". No, ORIC will not dispose of household rubbish, but it will purge those discarded strings and leave only strings currently owned by variables in the area we were just talking about. Garbage Collection can be forced at a desired point in your program by using the FRE(0) function at that point.

Arrays are yet another kettle of fish. In Zero Page live two more pointers:

1) Pointer to the Start of the Array Variables (SOAP)

2) Pointer to the End of the Array Variables (EOAP)

These two pointers reside at $9E and $A0 respectively. Of course, the amount of storage allocated to an Array depends on its Dimensions, and also on the Size of each of these Dimensions. However, the foundation of an Array variable is the same no matter what the size, or dimensions:



The first two bytes contain the Array's name, following the same convention that normal variables do. i.e. String Arrays will have the MSB of the second character set.

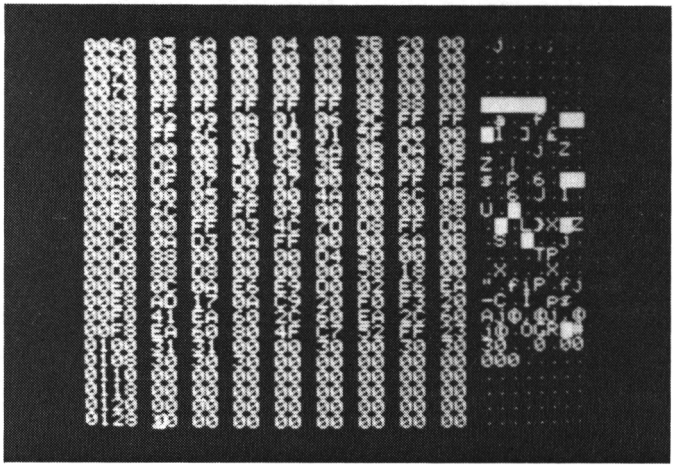
The Offset contained in bytes 3 & 4 can be added to the address of the Array in order to quickly point to the next Array variable following this one. This assists ORIC when it has to look for an Array variable mentioned in a Basic statement.

The next 2 bytes (5 & 6), say how many Dimensions the Array has. For instance, if you defined the Array A1(4,7) then it would have 2 dimensions. This would be stored as $02 & $00 in bytes 5 & 6 respectively.

Next come the Sizes of the individual Dimensions, using 2 bytes to keep the Size of each Dimension. ORIC keeps the Sizes in the opposite order to what they were defined in. ORIC also adds 1 to all of the sizes because you can use Elements 0 to n, giving n+1 Elements. i.e. A1(4,7) would give $08,$00 in bytes 7,8 and $05,$00 in bytes 9,10.

Following this definition part of the Array, are the particular Elements of the Array. The number of these can be obtained by multiplying the Sizes of the Dimensions together. i.e. For A1(4,7) there would be 40 Elements ((4+1)*(7+1)). Each of these Elements has the same structure as a String or Floating Point variable, except that Elements do not have name fields.

Machine Code Monitor



This program enables you to examine the contents of any memory location in ORIC, whether it be in ROM, or RAM. This includes : program area; variables; character sets; zero page ($0000-$00FF); etc, in fact ANY location from $0000 to $FFFF.

Although it isn't such a great program to use for keying a lot of data into memory, it is still quite useful for examining any part of memory and modifying it.

To use, you firstly give "Machine" the address of the "window" of memory you want to look at, and "Machine" will dump a screen of memory. Each line shows the contents of 8 memory locations in both Hex, on the left, and their ASCII character equivalents on the right. You can use the cursor control keys (up arrow, left arrow, down arrow, right arrow) to select the location you wish to change, or even to scroll the "window" up or down. To change the location being pointed at, just key in the new Hex value. If you make a mistake, you will be "ZAP"ped. Also if you try to make a change, which does not work (say you tried to change a location in ROM), you will be "SHOT".

STRUCTURE

FUNCTION	LINE(s)
Introduce program	4 - 50
Initialize Vertical & Horizontal position of cursor	60
Get HEX address from user (routine 1000)	70
Correct address to prevent "wrap-around" ($FFFF - $0000 or vice versa)	75
Turn cursor OFF	77
Print a screen full of memory contents	80
Turn cursor ON	85
Put cursor at Top Left Hand Corner of Dump (1st location displayed)	90 - 200
Get a Keypress	203
If Key was Return then END program (990)	210
If Key was left arrow then 300	230
If Key was right arrow then 400	240
If Key was down arrow then 420	250
If Key was up arrow then 320	260
Check for Alteration of Location (600)	270
If Key is not a Valid alteration then ZAP user	280
GOTO 200	290
Alter cursor co-ord 1 left : If cursor has gone too far left then make it go to the far right and then...	300 - 310
Alter cursor co-ord 1 up : If resultant address is still bigger than smallest address on the screen then 200	320 - 322
Prevent "wrap-around" of address by checking for negative address. If Neg. then address =0, put cursor back where it was and GOTO 322	325
Scroll page down to leave room for new Line of Info. Print Info for new address. Remember this address is the lowest	

displayed address. Alter the End address displayed because the last line has been pushed off the screen. GOTO 200	327 - 350
Shift cursor 1 Right. If cursor has gone too far then put it Far Left and...	400 - 410
Alter cursor 1 down : If resultant address is still less than or equal to end address then GOTO 200	420 - 422
Prevent "Wrap-around" of address by checking for greater than $FFF8, If greater than then Address =$FFF8, put cursor back where it was and GOTO 422	425
Scroll page up to leave room for new Line of Info. Print info for new address. Remember this address as the highest displayed address. Alter the starting address value because the first line on the screen has been pushed off. GOTO 200	427 - 450
Accept input from user (up to 2 characters). If ANY are not valid Hex #'s then set K less than 0 otherwise place valid Hex # at current location and return K greater than or equal to 0.	600 - 690
END	990 - 999
Input a Hex # (address) from the user and handle error conditions : (A) too many characters (B) illegal Hex Digits Convert Hex # to decimal and make it MOD(8) so that it will point to the start of a line	1000 - 1090
PRINT LINE OF INFORMATION	2000
Position Cursor	
Print address of this line	2005
Print Hex contents of this line as Hex numbers	2020
Print ASCII equivalents of memory contents	2030 - 2050
RETURN	2060
POSITION CURSOR	3000 - 3090
Validate Character into a Hex Digit	4000 - 4050
Input a Character from a Keypress	5000 - 5010

```
  1 REM * MACHINE CODE MONITOR *
  4 EXPLODE
  5 CLS
  6 INK 7
  7 FORJ=1TO20
  8 FOR I=7TO 0 STEP-1 : PAPERI : NEXT
  9 NEXTJ
 50 PRINT"MACHINE CODE MONITOR" : PRINT : PRINT
 60 VP=1 : HP=0
 70 GOSUB1000 : CLS
 75 SP=A : EP=A+25*8 : IF EP>#FFF8THEN
    SP=SP-(EP-#FFF8) : EP=#FFF8
 77 POKE 618 , PEEK(618)AND254
 80 FORA=SP TO EP STEP 8 : GOSUB2000 : VP=VP+1  :
    NEXT
 85 POKE 618 , PEEK(618)OR1
 90 A=SP : VP=1 : HP=0
200 X=HP*3+7 : Y=VP : GOSUB 3000
203 GOSUB5000
210 IF K=13 THEN 990
230 IF K=8 THEN 300
240 IF K=9 THEN 400
250 IF K=10 THEN 420
260 IF K=11 THEN 320
270 GOSUB600
280 IF K<0 THEN ZAP
290 GOTO 200
300 HP=HP-1 : IF HP>=0 THEN 200
310 HP=7
320 A=A-8 : VP=VP-1
322 IF A>=SP THEN 200
325 IF A<0 THEN A=0 : VP=1 : GOTO322
327 PRINTCHR$(11);
330 VP=VP+1 : GOSUB2000 : SP=SP-8
350 EP=EP-8 : GOTO200
400 HP=HP+1 : IF HP<8 THEN 200
410 HP=0
420 A=A+8 : VP=VP+1
422 IF A<=EP THEN 200
425 IF A>#FFF8 THEN A=#FFF8 : VP=VP-1 : GOTO422
427 PRINTCHR$(10);CHR$(10);
430 VP=VP-1 : GOSUB2000 : EP=EP+8
450 SP=SP+8 : GOTO200
600 GOSUB4000 : IF K<0 THEN RETURN
605 V=K
610 GOSUB5000
620 GOSUB4000 : IF K<0 THEN RETURN
630 V=V*16+K : POKEA+HP , V
640 PRINTRIGHT$("00"+MID$(HEX$(PEEK(A+HP)) , 2) ,
    2);
650 IF PEEK(A+HP)<>V THEN SHOOT
690 RETURN
990 POKE618 , PEEK(618)OR1
995 CLS
```

```
 999 END
1000 GOTO 1005
1002 SHOOT : PRINT"ERROR! - Illegal Hex Digit!"
1005 PRINT"Enter the Starting Address (in HEX)" :
     INPUT A$
1010 IF LEN(A$)>4 THENSHOOT : PRINT"ERROR! - Address
     too large!" : GOTO 1000
1020 P=1
1025 FOR I=LEN(A$) TO 1 STEP -1
1030 K$=MID$(A$ , I , 1) : K=ASC(K$)
1040 GOSUB 4000 : IFK<0THEN1002
1050 A=A+P*K : P=P*16
1060 NEXTI
1065 Q=FRE(0) : A=8*INT(A/8)
1070 RETURN
2000 Y=VP : X=2 : GOSUB 3000
2005 PRINTRIGHT$("0000"+MID$(HEX$(A) , 2) , 4);" ";
2010 FOR J=0TO7
2015 PRINTRIGHT$("00"+MID$(HEX$(PEEK(A+J)) , 2) ,
     2);" ";
2020 NEXTJ
2030 FOR J=0TO7
2035 V=PEEK(A+J) : IFV<32 THEN V=46
2040 IF V>127 AND V<160 THEN V=174
2045 PRINTCHR$(V);
2050 NEXTJ
2055 PRINT
2060 RETURN
3000 A1=PEEK(616)*40+PEEK(617)+48000
3020 POKE 616 , Y
3025 DOKE 18 , Y*40+48000 : POKE 617 , X
3030 POKE A1 , PEEK(A1)AND127
3090 RETURN
4000 K=K-48 : IF K<10 THEN 4050
4010 K=K-17 : IF K<0 THEN 4050
4020 K=K+10 : IF K>15 THEN K=-K
4050 RETURN
5000 K$=KEY$ : IF LEN(K$)=0 THEN 5000
5005 K=ASC(K$)
5010 RETURN
```

Renumber

Occasionally whilst developing a Basic program on ORIC, you will find yourself unable to insert a new block of code between two lines, because you didn't leave enough room for it. At other times you may just find your program full of bizarre line numbers. Well don't fret. This program will allow the user to renumber his Basic program so that : (1) new lines may be inserted where previously there was not room; and (2) to generally "neaten" the program.

This Renumber is not without its limitations, notably :

(1) It will only renumber the entire program, not selected portions of it.

(2) You cannot use line numbers greater than 64000 in your original program, as Renumber will use these whilst producing a completely Renumbered version of your program.

However, Renumber will handle all: GOTO; GOSUB; THEN; and ELSE constructs as well as the more complicated ON GOTO/GOSUB cases, is very quick, and allows the choice of different New Starting Lines and Increments. To instal Renumber in the System, after powering up, enter the following :

```
HIMEM #99FF
CLOAD"RENUMBER"
```

Renumber should then remain part of the system until another HIMEM. To use it, enter :

```
DOKE #400, (new starting line of program)
DOKE #402, (increment)
CALL #9A00
```

EXAMPLE:

```
DOKE #400,(100)
DOKE #402,(10)
CALL #9A00
```

This will start your program at line 100 and then increment every line by ten.

This program is complicated, so if you don't know a lot about Basic then read the "ORIC-1's Basic" section, or else you'll be behind the eight-ball. Anyway, Renumber works as follows :

1) Although you wouldn't notice from looking at it, Renumber actually renumbers the program twice in producing the finished product. This comes about because of Renumber's inability to DIRECTLY renumber a program to the same range of line numbers that it originally used. So, it first renumbers the program to 64001 and then back to the selected starting line# and increments.

2) Each one of the individual Renumbers starts renumbering from the End of the Program and works its way down to the Start. After it has renumbered the First Line of the program, it has finished.

3) To renumber a particular line, the following is done :

a) The Original Line No is changed to the new one;

b) A search is made through the program for a reference to the original line number. If found, then the new line number is placed there instead. If the two line numbers are not equal in LENGTH then the program is shifted about to accommodate the new line number;

c) The Link Addresses are corrected, along with EOPP, for any shifting of the program.

```
ORIC RENUMBER   ORIC RENUMBER

0000:            2         MSB  OFF
0000:            3 ***************************************************************
0000:            4 *                                                             *
0000:            5 *   PROGRAM I.D.: RENUMBER                                    *
0000:            6 *                                                             *
0000:            7 *   AUTHOR: CRAIG MC FARLANE                                  *
0000:            8 *                                                             *
0000:            9 *   MACHINE: THE ORIC-1                                       *
0000:           10 *                                                             *
0000:           11 *   FUNCTION:                                                 *
0000:           12 *                                                             *
0000:           13 *   THIS PROGRAM WILL RENUMBER ALL LINES IN A BASIC           *
0000:           14 *   PROGRAM. HANDLING ALL REFERENCES TO THOSE LINES           *
0000:           15 *   INCLUDING: GOTO; GOSUB; THEN; ELSE; AND ON GOTO/GOSUB.    *
0000:           16 *      THE PROGRAM WILL THEN START AT A SPECIFIED LINE#,      *
0000:           17 *   AND EACH LINE# AFTER WILL INCREASE BY A SET INCREMENT.    *
0000:           18 *                                                             *
0000:           19 ***************************************************************
0000:           21 ***********************************
0000:           22 *
0000:           23 *   PAGE ZERO
0000:           24 *

0000:           26         DSECT

0040:           28         ORG  $40
0040:           29 START   DS   2
0042:           30 END     DS   2
0044:           31 DEST    DS   2
0046:           32 POINT1  DS   2
0048:           33 POINT2  DS   2

009A:           35 SOPP    EQU  $9A
009C:           36 EOPP    EQU  $9C

0400:           38         ORG  $400

0400:           40 NSTART  DS   2
0402:           41 NINC    DS   2
0404:           42 ADJUST  DS   2
0406:           43 TEMPY   DS   1
0407:           44 LASTA   DS   2
0409:           45 SRCHL   DS   2
040B:           46 SRCHLC  DS   1
040C:           47 SRCHLS  DS   5
0411:           48 NEWL    DS   2
0413:           49 NEWLC   DS   1
0414:           50 NEWLS   DS   5
0419:           51 OBJLC   DS   1
041A:           52 OBJLS   DS   5

0000:           54         DEND

0097:           56 GOTO    EQU  $97
009B:           57 GOSUB   EQU  $9B
```

```
ORIC RENUMBER   ORIC RENUMBER

00C9:           58 THEN    EQU  $C9
00C8:           59 ELSE    EQU  $C8
00B4:           60 ON      EQU  $B4
0000:           61 *
0020:           62 SPACE   EQU  $20

0000:           64 ************************************
0000:           65 *
0000:           66 *   PROGRAM
0000:           67 *


----- NEXT OBJECT FILE NAME IS ORIC RENUMBER.OBJ0
9A00:           69         ORG  $9A00

9A00:AD 00 04   71 RENUM   LDA  NSTART    SAVE STARTING LINE# FOR LATER
9A03:48         72         PHA
9A04:AD 01 04   73         LDA  NSTART+1
9A07:48         74         PHA
9A08:AD 02 04   75         LDA  NINC       SAVE INCREMENT
9A0B:48         76         PHA
9A0C:AD 03 04   77         LDA  NINC+1
9A0F:48         78         PHA
9A10:           79 *
9A10:A5 9A      80         LDA  SOPP       POINT TO START OF PROGRAM
9A12:85 46      81         STA  POINT1
9A14:A5 9B      82         LDA  SOPP+1
9A16:85 47      83         STA  POINT1+1
9A18:           84 *
9A18:A0 00      85         LDY  #0         IS THERE A PROGRAM AT ALL?
9A1A:B1 46      86         LDA  (POINT1),Y
9A1C:D0 05      87         BNE  RENUM1
9A1E:C8         88         INY
9A1F:B1 46      89         LDA  (POINT1),Y
9A21:F0 2B      90         BEQ  RENUM9     NO? THEN SKIP AND END
9A23:           91 *
9A23:A9 01      92 RENUM1  LDA  #$01       RENUMBER FROM 64001
9A25:8D 00 04   93         STA  NSTART
9A28:A9 FA      94         LDA  #$FA
9A2A:8D 01 04   95         STA  NSTART+1
9A2D:           96 *
9A2D:A9 01      97         LDA  #1         INCREMENT OF 1
9A2F:8D 02 04   98         STA  NINC
9A32:A9 00      99         LDA  #0
9A34:8D 03 04  100         STA  NINC+1
9A37:          101 *
9A37:20 53 9A  102         JSR  REN000     DO INTERMEDIATE RENUMBER
9A3A:          103 *
9A3A:68        104         PLA             PUT BACK START & INCREMENT
9A3B:8D 03 04  105         STA  NINC+1
9A3E:68        106         PLA
9A3F:8D 02 04  107         STA  NINC
9A42:68        108         PLA
9A43:8D 01 04  109         STA  NSTART+1
9A46:68        110         PLA
```

```
ORIC RENUMBER   ORIC RENUMBER

9A47:8D 00 04  111         STA  NSTART
9A4A:          112 *
9A4A:20 53 9A  113         JSR  REN000    DO FINAL RENUMBER
9A4D:          114 *
9A4D:60        115         RTS

9A4E:68        117 RENUM9  PLA
9A4F:68        118         PLA
9A50:68        119         PLA
9A51:68        120         PLA
9A52:60        121         RTS

9A53:D8        123 REN000  CLD            CLEAR THAT DECIMAL FLAG
9A54:          124 *
9A54:A9 00     125         LDA  #0        NO LINES HAVE BEEN RENUMBERED YET
9A56:8D 07 04  126         STA  LASTA
9A59:8D 08 04  127         STA  LASTA+1
9A5C:          128 *
9A5C:A5 9A     129 REN005  LDA  SOPP      POINT TO START OF PROGRAM
9A5E:85 46     130         STA  POINT1
9A60:A5 9B     131         LDA  SOPP+1
9A62:85 47     132         STA  POINT1+1
9A64:          133 *
9A64:38        134 REN007  SEC            INITIALISE NEWLINE#
9A65:AD 00 04  135         LDA  NSTART
9A68:ED 02 04  136         SBC  NINC
9A6B:8D 11 04  137         STA  NEWL
9A6E:AD 01 04  138         LDA  NSTART+1
9A71:ED 03 04  139         SBC  NINC+1
9A74:8D 12 04  140         STA  NEWL+1
9A77:          141 *
9A77:A0 00     142 REN010  LDY  #0        LOOK AT CURRENT LINE
9A79:B1 46     143         LDA  (POINT1),Y
9A7B:AA        144         TAX
9A7C:C8        145         INY
9A7D:B1 46     146         LDA  (POINT1),Y
9A7F:A8        147         TAY            IS THIS THE LAST LINE OF PROGRAM
9A80:D0 04     148         BNE  REN015
9A82:E0 00     149         CPX  #0
9A84:F0 30     150         BEQ  REN030    YES? THEN SKIP
9A86:          151 *
9A86:A5 46     152 REN015  LDA  POINT1    ELSE HAVE WE ALREADY RENUMBERED
9A88:CD 07 04  153         CMP  LASTA     THE NEXT LINE?
9A8B:D0 07     154         BNE  REN020
9A8D:A5 47     155         LDA  POINT1+1
9A8F:CD 08 04  156         CMP  LASTA+1
9A92:F0 22     157         BEQ  REN030    YES? THEN SKIP
9A94:          158 *
9A94:18        159 REN020  CLC            NEWLINE#=NEWLINE#+NINC
9A95:AD 11 04  160         LDA  NEWL
9A98:6D 02 04  161         ADC  NINC
9A9B:8D 11 04  162         STA  NEWL
9A9E:AD 12 04  163         LDA  NEWL+1
9AA1:6D 03 04  164         ADC  NINC+1
9AA4:8D 12 04  165         STA  NEWL+1
```

```
9AA7:          166 *
9AA7:A5 46     167         LDA  POINT1    REMEMBER THIS CURRENT LINE
9AA9:85 48     168         STA  POINT2
9AAB:A5 47     169         LDA  POINT1+1
9AAD:85 49     170         STA  POINT2+1
9AAF:          171 *
9AAF:84 47     172         STY  POINT1+1  AND POINT TO NEXT ONE
9AB1:86 46     173         STX  POINT1
9AB3:4C 77 9A  174         JMP  REN010    LOOP BACK
9AB6:          175 *
9AB6:A0 02     176 REN030  LDY  #2        WE HAVE FOUND A LINE TO BE RENUMBERED
9AB8:B1 48     177         LDA  (POINT2),Y REMEMBER THE ORIGINAL LINE#
9ABA:8D 09 04  178         STA  SRCHL
9ABD:AA        179         TAX
9ABE:C8        180         INY
9ABF:B1 48     181         LDA  (POINT2),Y
9AC1:8D 0A 04  182         STA  SRCHL+1
9AC4:          183 *
9AC4:20 C6 9C  184         JSR  PRNUM     CONVERT IT TO A ASCII DECIMAL REPRES.
9AC7:A2 05     185         LDX  #5
9AC9:BD 13 04  186 REN032  LDA  NEWLC,X
9ACC:9D 0B 04  187         STA  SRCHLC,X
9ACF:CA        188         DEX
9AD0:10 F7     189         BPL  REN032
9AD2:          190 *
9AD2:A0 02     191         LDY  #2        REPLACE ORIGINAL# WITH NEW#
9AD4:AD 11 04  192         LDA  NEWL
9AD7:91 48     193         STA  (POINT2),Y
9AD9:AA        194         TAX
9ADA:C8        195         INY
9ADB:AD 12 04  196         LDA  NEWL+1
9ADE:91 48     197         STA  (POINT2),Y
9AE0:          198 *
9AE0:20 C6 9C  199         JSR  PRNUM     CONVERT NEW# TO ASCII DECIMAL
9AE3:          200 *
9AE3:A5 9A     201         LDA  SOPP      POINT TO START OF PROGRAM
9AE5:85 46     202         STA  POINT1
9AE7:A5 9B     203         LDA  SOPP+1
9AE9:85 47     204         STA  POINT1+1
9AEB:          205 *
9AEB:A5 48     206         LDA  POINT2    REMEMBER THIS LINE AS RENUMBERED
9AED:8D 07 04  207         STA  LASTA
9AF0:A5 49     208         LDA  POINT2+1
9AF2:8D 08 04  209         STA  LASTA+1
9AF5:          210 *
9AF5:A0 04     211 REN035  LDY  #4
9AF7:          212 *
9AF7:B1 46     213 REN040  LDA  (POINT1),Y END OF RECORD?
9AF9:F0 23     214         BEQ  REN065    YES? THEN SKIP
9AFB:10 1D     215         BPL  REN060    IF NOT A TOKEN THEN SKIP
9AFD:C9 97     216         CMP  #GOTO     IS IT A GOTO/GOSUB/THEN/ELSE?
9AFF:F0 0C     217         BEQ  REN045
9B01:C9 9B     218         CMP  #GOSUB
9B03:F0 08     219         BEQ  REN045
9B05:C9 C8     220         CMP  #ELSE
```

```
9B07:F0 04     221         BEQ  REN045
9B09:C9 C9     222         CMP  #THEN
9B0B:D0 06     223         BNE  REN050    NO? THEN SKIP
9B0D:20 44 9B  224 REN045  JSR  REN100    ELSE HANDLE GOTO ETC...
9B10:4C 1A 9B  225         JMP  REN060
9B13:          226 *
9B13:C9 B4     227 REN050  CMP  #ON       IS IT AN ON GOTO/GOSUB?
9B15:D0 03     228         BNE  REN060    NO? THEN SKIP
9B17:20 17 9C  229         JSR  REN200    ELSE HANDLE ON ETC....
9B1A:          230 *
9B1A:C8        231 REN060  INY            LOOK AT NEXT CHARACTER IN RECORD
9B1B:4C F7 9A  232         JMP  REN040
9B1E:          233 *
9B1E:A0 00     234 REN065  LDY  #0        POINT TO NEXT RECORD
9B20:B1 46     235         LDA  (POINT1),Y
9B22:AA        236         TAX
9B23:C8        237         INY
9B24:B1 46     238         LDA  (POINT1),Y
9B26:85 47     239         STA  POINT1+1
9B28:86 46     240         STX  POINT1
9B2A:          241 *
9B2A:D0 C9     242         BNE  REN035    END OF PROGRAM?
9B2C:E0 00     243         CPX  #0
9B2E:D0 C5     244         BNE  REN035    NO? THEN LOOP BACK
9B30:          245 *
9B30:AD 11 04  246         LDA  NEWL      ELSE HAVE WE FINISHED RENUMBER?
9B33:CD 00 04  247         CMP  NSTART
9B36:D0 08     248         BNE  REN070
9B38:AD 12 04  249         LDA  NEWL+1
9B3B:CD 01 04  250         CMP  NSTART+1
9B3E:F0 03     251         BEQ  REN099
9B40:4C 5C 9A  252 REN070  JMP  REN005    NO? THEN LOOP BACK
9B43:          253 *
9B43:60        254 REN099  RTS

9B44:          256 ##################################
9B44:          257 *
9B44:          258 *   GOTO/GOSUB/THEN/ELSE HANDLER
9B44:          259 *

9B44:C8        261 REN100  INY            POINT PAST TOKEN
9B45:20 2E 9D  262         JSR  SKPBLK    SKIP ANY SPACES
9B48:8C 06 04  263         STY  TEMPY     THIS SHOULD BE START OF NUMBER
9B4B:20 9C 9C  264         JSR  PARNUM    GET NUMBER
9B4E:90 04     265         BCC  REN120    O.K.? THEN SKIP
9B50:88        266         DEY            ELSE NO NUMBER! BACKSTEP & END
9B51:4C 16 9C  267 REN110  JMP  REN199
9B54:          268 *
9B54:AE 0B 04  269 REN120  LDX  SRCHLC    IS THIS LINE# AS LONG AS SEARCH#
9B57:EC 19 04  270         CPX  OBJLC
9B5A:D0 F5     271         BNE  REN110    NO? THEN END
9B5C:          272 *
9B5C:CA        273 REN130  DEX            ELSE SEE IF DIGITS MATCH UP
9B5D:30 0A     274         BMI  REN140
9B5F:BD 0C 04  275         LDA  SRCHLS,X
```

```
9B62:DD 1A 04  276         CMP  OBJLS,X
9B65:D0 EA     277         BNE  REN110    NOT THE SAME? THEN END
9B67:F0 F3     278         BEQ  REN130    KEEP CHECKING UNTIL OUT OF DIGITS
9B69:          279 *
9B69:38        280 REN140  SEC            CALCULATE DIFFERENCE IN LENGTH
9B6A:A2 00     281         LDX  #0        BETWEEN TWO NUMBERS
9B6C:AD 13 04  282         LDA  NEWLC
9B6F:ED 19 04  283         SBC  OBJLC
9B72:8D 04 04  284         STA  ADJUST
9B75:10 02     285         BPL  REN145
9B77:A2 FF     286         LDX  #$FF
9B79:8E 05 04  287 REN145  STX  ADJUST+1
9B7C:          288 *
9B7C:18        289         CLC            MOVE MEMORY TO ACCOMODATE NEW LINE#
9B7D:98        290         TYA
9B7E:65 46     291         ADC  POINT1
9B80:85 40     292         STA  START
9B82:A5 47     293         LDA  POINT1+1
9B84:69 00     294         ADC  #0
9B86:85 41     295         STA  START+1
9B88:          296 *
9B88:18        297         CLC
9B89:A5 40     298         LDA  START
9B8B:6D 04 04  299         ADC  ADJUST
9B8E:85 44     300         STA  DEST
9B90:A5 41     301         LDA  START+1
9B92:6D 05 04  302         ADC  ADJUST+1
9B95:85 45     303         STA  DEST+1
9B97:          304 *
 9B97:A5 9C    305         LDA  EOPP
9B99:85 42     306         STA  END
9B9B:A5 9D     307         LDA  EOPP+1
9B9D:85 43     308         STA  END+1
9B9F:          309 *
9B9F:20 35 9C  310         JSR  MOVE
9BA2:          311 *
9BA2:A5 46     312         LDA  POINT1    MODIFY LINK ADDRESSES FROM
9BA4:85 48     313         STA  POINT2    THIS LINE ON
9BA6:A5 47     314         LDA  POINT1+1
9BA8:85 49     315         STA  POINT2+1
9BAA:          316 *
9BAA:A0 00     317 REN150  LDY  #0
9BAC:18        318         CLC
9BAD:B1 48     319         LDA  (POINT2),Y
9BAF:08        320         PHP
9BB0:6D 04 04  321         ADC  ADJUST
9BB3:91 48     322         STA  (POINT2),Y
9BB5:AA        323         TAX
9BB6:C8        324         INY
9BB7:B1 48     325         LDA  (POINT2),Y
9BB9:08        326         PHP
9BBA:6D 05 04  327         ADC  ADJUST+1
9BBD:91 48     328         STA  (POINT2),Y
9BBF:          329 *
9BBF:28        330         PLP            WAS THAT THE EOP MARKER ($0000)?
```

```
ORIC RENUMBER   ORIC RENUMBER

9BC0:F0 08      331         BEQ  REN155
9BC2:28         332         PLP           NO? THEN LOOK AT NEXT LINE
9BC3:85 49      333 REN153  STA  POINT2+1
9BC5:86 48      334         STX  POINT2
9BC7:4C AA 9B   335         JMP  REN150
9BCA:28         336 REN155  PLP           ELSE DE-MODIFY EOP MARKER
9BCB:D0 F6      337         BNE  REN153
9BCD:           338 *
9BCD:A9 00      339         LDA  #0
9BCF:91 48      340         STA  (POINT2),Y
9BD1:88         341         DEY
9BD2:91 48      342         STA  (POINT2),Y
9BD4:           343 *
9BD4:A5 47      344         LDA  POINT1+1  MODIFY ADDRESS OF LAST LINE
9BD6:CD 08 04   345         CMP  LASTA+1   TO BE RENUMBERED IF THIS
9BD9:90 07      346         BCC  REN160    LINE IS BEFORE IT
9BDB:A5 46      347         LDA  POINT1
9BDD:CD 07 04   348         CMP  LASTA
9BE0:B0 13      349         BCS  REN170
9BE2:           350 *
9BE2:18         351 REN160  CLC
9BE3:AD 07 04   352         LDA  LASTA
9BE6:6D 04 04   353         ADC  ADJUST
9BE9:8D 07 04   354         STA  LASTA
9BEC:AD 08 04   355         LDA  LASTA+1
9BEF:6D 05 04   356         ADC  ADJUST+1
9BF2:8D 08 04   357         STA  LASTA+1
9BF5:           358 *
9BF5:18         359 REN170  CLC           CALCULATE NEW EOPP
9BF6:A5 9C      360         LDA  EOPP
9BF8:6D 04 04   361         ADC  ADJUST
9BFB:85 9C      362         STA  EOPP
9BFD:A5 9D      363         LDA  EOPP+1
9BFF:6D 05 04   364         ADC  ADJUST+1
9C02:85 9D      365         STA  EOPP+1
9C04:           366 *
9C04:AC 06 04   367         LDY  TEMPY    PUT NEW LINE# IN
9C07:A2 00      368         LDX  #0
9C09:           369 *
9C09:BD 14 04   370 REN180  LDA  NEWLS,X
9C0C:91 46      371         STA  (POINT1),Y
9C0E:C8         372         INY
9C0F:E8         373         INX
9C10:EC 13 04   374         CPX  NEWLC
9C13:D0 F4      375         BNE  REN180
9C15:88         376         DEY
9C16:           377 *
9C16:60         378 REN199  RTS

9C17:           380 ********************************
9C17:           381 *
9C17:           382 *   ON GOTO/GOSUB HANDLER
9C17:           383 *
```

```
9C17:C8        385 REN200  INY
9C18:B1 46     386         LDA  (POINT1),Y LOOK FOR A GOTO/GOSUB
9C1A:D0 02     387         BNE  REN205     END OF LINE?
9C1C:88        388         DEY             YES? THEN END
9C1D:60        389         RTS
9C1E:          390 *
9C1E:C9 97     391 REN205  CMP  #GOTO
9C20:F0 04     392         BEQ  REN210
9C22:C9 9B     393         CMP  #GOSUB
9C24:D0 F1     394         BNE  REN200
9C26:          395 *
9C26:20 44 9B  396 REN210  JSR  REN100     RENUMBER THIS LINE REFERENCE
9C29:C8        397         INY
9C2A:20 2E 9D  398         JSR  SKPBLK     SKIP ANY BLANKS
9C2D:B1 46     399         LDA  (POINT1),Y ANY MORE REFERENCES
9C2F:C9 2C     400         CMP  #',
9C31:F0 F3     401         BEQ  REN210     YES? THEN LOOP BACK
9C33:88        402         DEY
9C34:          403 *
9C34:60        404 REN299  RTS

9C35:          406 ************************************
9C35:          407 *
9C35:          408 *   MOVE MEMORY
9C35:          409 *

9C35:98        411 MOVE    TYA             SAVE Y
9C36:48        412         PHA
9C37:A0 00     413         LDY  #0
9C39:          414 *
9C39:A5 41     415         LDA  START+1    IF START < DEST THEN SKIP
9C3B:C5 45     416         CMP  DEST+1
9C3D:90 20     417         BCC  MOVE05
9C3F:A5 40     418         LDA  START
9C41:C5 44     419         CMP  DEST
9C43:90 1A     420         BCC  MOVE05
9C45:          421 *
9C45:B1 40     422 MOVE01  LDA  (START),Y MOVE BYTE
9C47:91 44     423         STA  (DEST),Y
9C49:          424 *
9C49:E6 44     425         INC  DEST       NEXT LOCATION TO MOVE TO
9C4B:D0 02     426         BNE  MOVE02
9C4D:E6 45     427         INC  DEST+1
9C4F:20 91 9C  428 MOVE02  JSR  COMP       FINISHED?
9C52:D0 03     429         BNE  MOVE04     NO? THEN SKIP
9C54:          430 *
9C54:68        431 MOVE03  PLA             ELSE RESTORE Y & RETURN
9C55:A8        432         TAY
9C56:60        433         RTS
9C57:          434 *
9C57:E6 40     435 MOVE04  INC  START      NEXT LOCATION TO MOVE FROM
9C59:D0 EA     436         BNE  MOVE01
9C5B:E6 41     437         INC  START+1
9C5D:D0 E6     438         BNE  MOVE01
```

```
9C5F:          439 *
9C5F:          440 *
9C5F:          441 *
9C5F:38        442 MOVE05  SEC           MUST MOVE FROM TOP DOWN
9C60:A5 42     443         LDA  END
9C62:E5 40     444         SBC  START
9C64:AA        445         TAX
9C65:A5 43     446         LDA  END+1
9C67:E5 41     447         SBC  START+1
9C69:48        448         PHA
9C6A:18        449         CLC
9C6B:8A        450         TXA
9C6C:65 44     451         ADC  DEST
9C6E:85 44     452         STA  DEST
9C70:68        453         PLA
9C71:65 45     454         ADC  DEST+1
9C73:85 45     455         STA  DEST+1
9C75:          456 *
9C75:B1 42     457 MOVE06  LDA  (END),Y   MOVE BYTE
9C77:91 44     458         STA  (DEST),Y
9C79:          459 *
9C79:A5 44     460         LDA  DEST
9C7B:D0 02     461         BNE  MOVE07
9C7D:C6 45     462         DEC  DEST+1
9C7F:C6 44     463 MOVE07  DEC  DEST
9C81:20 91 9C  464         JSR  COMP      FINISHED
9C84:F0 CE     465         BEQ  MOVE03    YES? THEN END
9C86:A5 42     466         LDA  END
9C88:D0 02     467         BNE  MOVE08
9C8A:C6 43     468         DEC  END+1
9C8C:C6 42     469 MOVE08  DEC  END
9C8E:4C 75 9C  470         JMP  MOVE06
9C91:          471 *
9C91:          472 *
9C91:          473 *
9C91:A5 40     474 COMP    LDA  START     COMPARE START & END
9C93:C5 42     475         CMP  END
9C95:D0 04     476         BNE  COMP1
9C97:A5 41     477         LDA  START+1
9C99:C5 43     478         CMP  END+1
9C9B:60        479 COMP1   RTS


9C9C:          481 ********************************
9C9C:          482 *
9C9C:          483 *   PARNUM - GET ASCII DEC LINE# INTO BUFFER
9C9C:          484 *
9C9C:          485 * RETURNED: Y - POINTS TO LAST DIGIT IN LINE#
9C9C:          486 *           PS - C SET IF NO DIGITS
9C9C:          487 *

9C9C:A2 00     489 PARNUM  LDX  #0
9C9E:B1 46     490         LDA  (POINT1),Y
9CA0:20 22 9D  491         JSR  VALDIG
9CA3:B0 20     492         BCS  PARNUM9   NO NUMBER AT ALL? THEN END
```

```
ORIC RENUMBER   ORIC RENUMBER

9CA5:           493 *
9CA5:C8         494         INY
9CA6:9D 1A 04   495         STA  OBJLS,X   OTHERWISE PUT CHAR IN BUFFER
9CA9:E8         496         INX
9CAA:A9 04      497         LDA  #4        MAX OF 4 MORE CHARACTERS
9CAC:8D 19 04   498         STA  OBJLC
9CAF:           499 *
9CAF:B1 46      500 PARNUM1 LDA  (POINT1),Y GET NEXT CHARACTER
9CB1:20 22 9D   501         JSR  VALDIG
9CB4:B0 0A      502         BCS  PARNUM8   NOT A NUMBER? THEN END
9CB6:C8         503         INY
9CB7:9D 1A 04   504         STA  OBJLS,X   ELSE STORE IN BUFFER
9CBA:E8         505         INX            NEXT BUFFER POSITION
9CBB:CE 19 04   506         DEC  OBJLC     MORE DIGITS TO GET?
9CBE:D0 EF      507         BNE  PARNUM1   YES? THEN LOOP
9CC0:           508 *
9CC0:8E 19 04   509 PARNUM8 STX  OBJLC     SAVE THE #DIGITS GOTTEN
9CC3:88         510         DEY            Y POINTS TO LAST DIGIT OF LINE#
9CC4:18         511         CLC
9CC5:           512 *
9CC5:60         513 PARNUM9 RTS

9CC6:           515 ***********************************
9CC6:           516 *
9CC6:           517 *   PRNUM - PRINT # INTO BUFFER
9CC6:           518 *

9CC6:8D 05 04   520 PRNUM   STA  ADJUST+1
9CC9:8E 04 04   521         STX  ADJUST
9CCC:           522 *
9CCC:A2 00      523         LDX  #0
9CCE:8E 13 04   524         STX  NEWLC
9CD1:A0 00      525         LDY  #0
9CD3:A9 05      526         LDA  #5
9CD5:8D 06 04   527         STA  TEMPY
9CD8:           528 *
9CD8:A9 30      529 PRNUM1  LDA  #'0
9CDA:9D 14 04   530         STA  NEWLS,X
9CDD:           531 *
9CDD:38         532 PRNUM2  SEC
9CDE:AD 04 04   533         LDA  ADJUST
9CE1:F9 37 9D   534         SBC  TAB,Y
9CE4:8D 04 04   535         STA  ADJUST
9CE7:AD 05 04   536         LDA  ADJUST+1
9CEA:F9 38 9D   537         SBC  TAB+1,Y
9CED:8D 05 04   538         STA  ADJUST+1
9CF0:90 05      539         BCC  PRNUM3
9CF2:FE 14 04   540         INC  NEWLS,X
9CF5:D0 E6      541         BNE  PRNUM2
9CF7:           542 *
9CF7:AD 13 04   543 PRNUM3  LDA  NEWLC
9CFA:D0 07      544         BNE  PRNUM4
9CFC:BD 14 04   545         LDA  NEWLS,X
9CFF:C9 30      546         CMP  #'0
9D01:F0 04      547         BEQ  PRNUM5
```

```
9D03:          548 *
9D03:E8        549 PRNUM4  INX
9D04:EE 13 04  550         INC  NEWLC
9D07:          551 *
9D07:18        552 PRNUM5  CLC
9D08:AD 04 04  553         LDA  ADJUST
9D0B:79 37 9D  554         ADC  TAB,Y
9D0E:8D 04 04  555         STA  ADJUST
9D11:AD 05 04  556         LDA  ADJUST+1
9D14:79 38 9D  557         ADC  TAB+1,Y
9D17:8D 05 04  558         STA  ADJUST+1
9D1A:C8        559         INY
9D1B:C8        560         INY
9D1C:CE 06 04  561         DEC  TEMPY
9D1F:D0 B7     562         BNE  PRNUM1
9D21:          563 *
9D21:60        564         RTS

9D22:          566 ##################################
9D22:          567 *
9D22:          568 *   VALDIG
9D22:          569 *


9D22:C9 30     571 VALDIG  CMP  #'0       SET CARRY IF DIGIT NOT DECIMAL
9D24:90 06     572         BCC  VALDIG9
9D26:C9 3A     573         CMP¶ #'9+1
9D28:B0 02     574         BCS  VALDIG9
9D2A:18        575         CLC
9D2B:60        576         RTS
9D2C:          577 *
9D2C:38        578 VALDIG9 SEC
9D2D:60        579         RTS


9D2E:B1 46     581 SKPBLK  LDA  (POINT1),Y SKIP BLANKS
9D30:C8        582         INY
9D31:C9 20     583         CMP  #SPACE
9D33:F0 F9     584         BEQ  SKPBLK
9D35:88        585         DEY
9D36:60        586         RTS

9D37:10 27     588 TAB     DW   10000
9D39:E8 03     589         DW   1000
9D3B:64 00     590         DW   100
9D3D:0A 00     591         DW   10
9D3F:01 00     592         DW   1
```

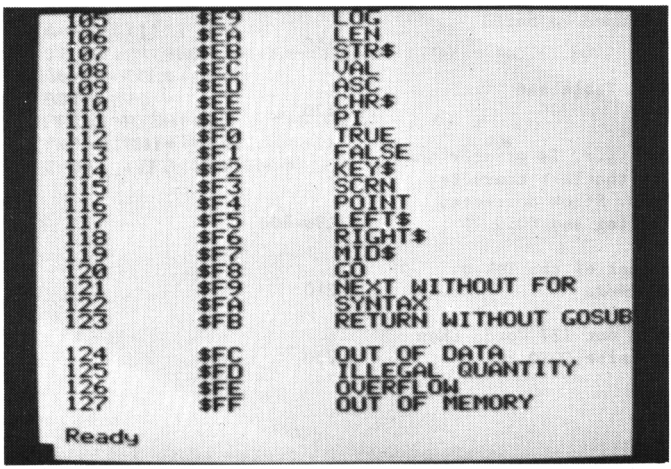
SYMBOL TABLE SORTED BY SYMBOL

0404 ADJUST	9C9B COMP1	9C91 COMP	44 DEST
C8 ELSE	42 END	9C EOPP	9B GOSUB
97 GOTO	0407 LASTA	9C45 MOVE01	9C4F MOVE02
9C54 MOVE03	9C57 MOVE04	9C5F MOVE05	9C75 MOVE06
9C35 MOVE	9C7F MOVE07	9C8C MOVE08	0411 NEWL
0413 NEWLC	0414 NEWLS	0402 NINC	0400 NSTART
0419 OBJLC	041A OBJLS	B4 ON	9CAF PARNUM1
9CC5 PARNUM9	9C9C PARNUM	9CC0 PARNUM8	46 POINT1
48 POINT2	9CD8 PRNUM1	9CDD PRNUM2	9D07 PRNUM5
9CC6 PRNUM	9CF7 PRNUM3	9D03 PRNUM4	9A53 REN000
9A5C REN005	?9A64 REN007	9A77 REN010	9A86 REN015
9A94 REN020	9AB6 REN030	9AC9 REN032	9AF5 REN035
9AF7 REN040	9B0D REN045	9B13 REN050	9B1A REN060
9B1E REN065	9B40 REN070	9B43 REN099	9B44 REN100
9B51 REN110	9B54 REN120	9B5C REN130	9B69 REN140
9B79 REN145	9BAA REN150	9BC3 REN153	9BCA REN155
9BE2 REN160	9BF5 REN170	9C09 REN180	9C16 REN199
9C17 REN200	9C1E REN205	9C26 REN210	?9C34 REN299
9A4E RENUM9	?9A00 RENUM	9A23 RENUM1	9D2E SKPBLK
9A SOPP	20 SPACE	0409 SRCHL	0408 SRCHLC
040C SRCHLS	40 START	9D37 TAB	0406 TEMPY
C9 THEN	9D22 VALDIG	9D2C VALDIG9	

SORTED BY ADDRESS

20 SPACE	40 START	42 END	44 DEST
46 POINT1	48 POINT2	97 GOTO	9A SOPP
9B GOSUB	9C EOPP	B4 ON	C8 ELSE
C9 THEN	0400 NSTART	0402 NINC	0404 ADJUST
0406 TEMPY	0407 LASTA	0409 SRCHL	040B SRCHLC
040C SRCHLS	0411 NEWL	0413 NEWLC	0414 NEWLS
0419 OBJLC	041A OBJLS	?9A00 RENUM	9A23 RENUM1
9A4E RENUM9	9A53 REN000	9A5C REN005	?9A64 REN007
9A77 REN010	9A86 REN015	9A94 REN020	9AB6 REN030
9AC9 REN032	9AF5 REN035	9AF7 REN040	9B0D REN045
9B13 REN050	9B1A REN060	9B1E REN065	9B40 REN070
9B43 REN099	9B44 REN100	9B51 REN110	9B54 REN120
9B5C REN130	9B69 REN140	9B79 REN145	9BAA REN150
9BC3 REN153	9BCA REN155	9BE2 REN160	9BF5 REN170
9C09 REN180	9C16 REN199	9C17 REN200	9C1E REN205
9C26 REN210	?9C34 REN299	9C35 MOVE	9C45 MOVE01
9C4F MOVE02	9C54 MOVE03	9C57 MOVE04	9C5F MOVE05
9C75 MOVE06	9C7F MOVE07	9C8C MOVE08	9C91 COMP
9C9B COMP1	9C9C PARNUM	9CAF PARNUM1	9CC0 PARNUM8
9CC5 PARNUM9	9CC6 PRNUM	9CD8 PRNUM1	9CDD PRNUM2
9CF7 PRNUM3	9D03 PRNUM4	9D07 PRNUM5	9D22 VALDIG
9D2C VALDIG9	9D2E SKPBLK	9D37 TAB	

Token Table



Token Table is a little program that will list out ORIC's Tokens and their respective Reserved Words. It does this by accessing a table in the Basic ROM area. This table contains all the Reserved Words in an order that corresponds to their Token numbers. The End of a Reserved Word in this table is indicated by the MSB of the last character being set i.e. the Reserved Word "END" will be entered in the table as : $45,$4E,$C4.

STRUCTURE	LINE(s)
Print Headings	120-140
Point to Start of Token Table Initialize Token # to 1st Token	150
Initialize String used to build Reserved Word	160
Get Character from Table and Increment pointer	170
IF the MSB is set (i.e. is greater than 127) THEN this is the last character of a Reserved Word. Stick corrected character onto String and GOTO 70	190-200
Print out the number of the Token, Token #, Reserved Word	210
IF we have printed out 127 Words then we have finished, else GOTO 20	220

```
100 REM * TOKEN TABLE *
110 REM
120 CLS
130 PRINT"NUMBER" , "TOKEN" , "RESERVED WORD"
140 PRINT
150 I=49386 : J=1
160 A$=""
170 A=PEEK(I) : I=I+1
180 IF A>127 THEN A$=A$+CHR$(A-128) : GOTO210
190 A$=A$+CHR$(A)
200 GOTO 170
210 PRINT RIGHT$("    "+STR$(J) , 4) ,  ,
    "$"+RIGHT$(HEX$(J+#80) , 2) ,  , A$
220 J=J+1 :IFJ<128 THEN 160
```

Block Line Deletion

The ORIC-1 does not have a Block Delete facility that enables the user to delete a range of lines from his Basic program. This program will allow the user to do just that. It will only work if both the Starting and Ending lines specified actually exist in the program. If not, no action will be taken. As no error messages are returned if either or both line numbers are wrong, it is a good idea to check by listing the relevant lines. To instal Blockdel, after powering up, enter the following :

```
HIMEM #9DFF
CLOAD"BLOCKDEL
```

Blockdel should then remain part of ORIC's works until the next HIMEM. To use it, enter :

```
DOKE #400, (first line of Block)
DOKE #402, (last line of Block)
CALL #9E00
```

To understand the operation of Block Delete, you should read the section titled ORIC-1's Basic. If you HAVE then read on. Block Delete works by doing the following :

1) Examine the following diagram of your typical Basic program :

```
                                        Start of Program
                                        (SOP)

                                        Line Before Line
                                        #1 (L1')
(L1) Line #1

                                        Code to be
                                        deleted

(L2) Line #2
                                        Line after line
                                        #2 (L2')

(EOP) End of Program
```

And pictured below is what you want the program to look like :

```
                                        Start of Program

Line before Line #1 (L1')
                                        Line after Line #2
                                        (L2')

End of Program
```

We achieve this simply by moving the Block of code from L2' → EOP down to where L1 was. This effectively wipes the code we wanted to delete and leaves the remaining code in its place. However, we are not finished yet... .

2) Because we have moved the Block of code L2' → EOP, the Link Addresses for that Block of code will be incorrect. To fix this, we subtract the distance (in bytes) moved, from the Link Addresses originally there, thus reforming the chain of Basic Statements. Finally...

3) The EOPP in zero page must be updated to point at the new EOP.

STRUCTURE

FUNCTION	LINE(s)
Define locations used by BLOCKDEL	17 - 38
Clear Decimal Flag so that all calculations are done in Hex and not BCD	47
Scan through program looking for Line #1. If not there then end program with NO action	49 - 73
Scan through program from line following Line #1 looking for Line #2. If not there then end program with NO action	75 - 101
Point to line following Line #2	103 - 109
Calculate how many bytes apart Line #1 and the line following Line #2 are. Then shift all lines after Line #2 down to where Line #1 was, effectively destroying Line #1-Line 2 inclusive	111 - 121
Calculate the new Link Addresses for all of those lines shifted down, calculating new EOPP as it goes	123 - 146
Add 2 to EOPP to give correct value	148 - 152
RETURN	153
Move Memory Routine	156 - 182

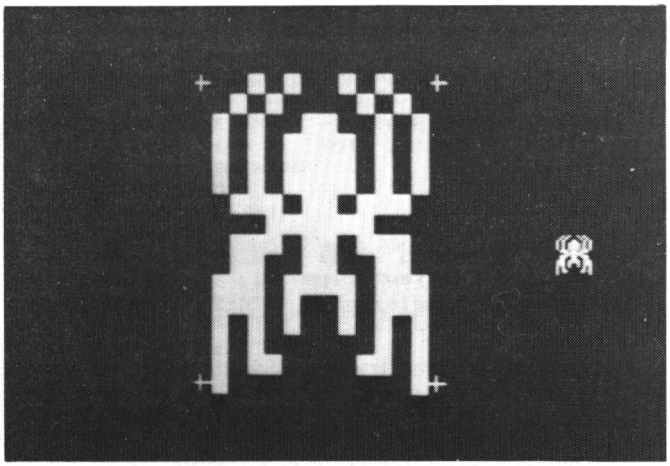
```
BLOCK DELETE    ORIC BLOCK DELETE PROGRAM

0000:            2 ************************************************************
0000:            3 *                                                          *
0000:            4 *   PROGRAM I.D.: ORIC BLOCK DELETE       DATE: 27/04/83   *
0000:            5 *                                                          *
0000:            6 *   AUTHOR: CRAIG MC FARLANE                               *
0000:            7 *                                                          *
0000:            8 *   MACHINE: THE ORIC-1                                    *
0000:            9 *                                                          *
0000:           10 *   FUNCTION:                                              *
0000:           11 *                                                          *
0000:           12 *      THIS PROGRAM WILL ALLOW THE USER TO DELETE A BLOCK  *
0000:           13 *   OF LINES FROM A BASIC PROGRAM.                         *
0000:           14 *                                                          *
0000:           15 *   TO INSTALL BLOCKDEL IN THE ORIC KEY IN FROM BASIC.     *
0000:           16 *                                                          *
0000:           17 *   HIMEM #9DFF                                            *
0000:           18 *   CLOAD "BLOCKDEL"                                       *
0000:           19 *                                                          *
0000:           20 *   BLOCKDEL SHOULD NOW BE LOADED. IT WILL STAY THERE      *
0000:           21 *   UNTIL A HIMEM, NEW, CLOAD OR A RESET IS PERFORMED.     *
0000:           22 *   TO USE BLOCKDEL, JUST KEY IN THE FOLLOWING:            *
0000:           23 *                                                          *
0000:           24 *   DOKE #400, STARTING LINE#                              *
0000:           25 *   DOKE #402, ENDING LINE#                                *
0000:           26 *   CALL #9E00                                             *
0000:           27 *                                                          *
0000:           28 ************************************************************

0000:           30 ************************************
0000:           31 *
0000:           32 *   ZERO PAGE
0000:           33 *

0000:           35          DSECT

0040:           37          ORG  $40        A HOPEFULLY FREE AREA OF ZERO PAGE

0040:           39 DEST     DS   2
0042:           40 START    DS   2
0044:           41 ADJUST   DS   2

009A:           43 SOPP     EQU  $9A        POINTER TO START OF BASIC PROGRAM
009C:           44 EOPP     EQU  $9C        POINTER TO END OF BASIC PROGRAM

0400:           46          ORG  $400

0400:           48 STARTL   DS   2          START OF BLOCK TO BE DELETED
0402:           49 ENDL     DS   2          END OF BLOCK

0000:           51          DEND

0000:           53 ************************************
0000:           54 *
0000:           55 *   MAIN PROGRAM
0000:           56 *
```

```
----- NEXT OBJECT FILE NAME IS BLOCK DELETE.OBJ0
9E00:           58          ORG  $9E00

9E00:D8         60 BDEL00   CLD             CLEAR THAT DAMN DECIMAL FLAG
9E01:           61 *
9E01:A5 9A      62          LDA  SOPP       START SCANNING FROM START OF PROGRAM
9E03:85 40      63          STA  DEST
9E05:A5 9B      64          LDA  SOPP+1
9E07:85 41      65          STA  DEST+1
9E09:           66 *
9E09:A0 02      67 BDEL05   LDY  #2          FIND STARTING LINE
9E0B:B1 40      68          LDA  (DEST),Y
9E0D:AA         69          TAX
9E0E:C8         70          INY
9E0F:B1 40      71          LDA  (DEST),Y   IS THIS THE END MARKER OF PROGRAM
9E11:D0 04      72          BNE  BDEL10     NO? THEN SKIP
9E13:E0 00      73          CPX  #0
9E15:F0 31      74          BEQ  BDEL28     OTHERWISE DO NOTHING BUT RETURN
9E17:EC 00 04   75 BDEL10   CPX  STARTL     IS THIS THE STARTING LINE?
9E1A:D0 05      76          BNE  BDEL15     NO? THEN POINT TO NEXT LINE
9E1C:CD 01 04   77          CMP  STARTL+1
9E1F:F0 0F      78          BEQ  BDEL20     OTHERWISE END SEARCH
9E21:A0 00      79 BDEL15   LDY  #0          POINT TO NEXT LINE
9E23:B1 40      80          LDA  (DEST),Y
9E25:AA         81          TAX
9E26:C8         82          INY
9E27:B1 40      83          LDA  (DEST),Y
9E29:85 41      84          STA  DEST+1
9E2B:86 40      85          STX  DEST
9E2D:4C 09 9E   86          JMP  BDEL05     LOOP BACK & KEEP LOOKING
9E30:           87 *
9E30:A0 00      88 BDEL20   LDY  #0          FIND ENDING LINE
9E32:B1 40      89          LDA  (DEST),Y   START LOOKING FROM LINE AFTER START
9E34:AA         90          TAX
9E35:C8         91          INY
9E36:B1 40      92          LDA  (DEST),Y
9E38:85 43      93          STA  START+1
9E3A:86 42      94          STX  START
9E3C:A0 02      95 BDEL25   LDY  #2
9E3E:B1 42      96          LDA  (START),Y
9E40:AA         97          TAX
9E41:C8         98          INY
9E42:B1 42      99          LDA  (START),Y IS THIS THE END MARKER OF PROGRAM
9E44:D0 04     100          BNE  BDEL30     NO? THEN SKIP
9E46:E0 00     101          CPX  #0
9E48:F0 69     102 BDEL28   BEQ  BDEL99     OTHERWISE RETURN & DO NOTHING
9E4A:EC 02 04  103 BDEL30   CPX  ENDL       IS THIS THE END LINE?
9E4D:D0 05     104          BNE  BDEL35     NO? THEN POINT TO NEXT LINE
9E4F:CD 03 04  105          CMP  ENDL+1
9E52:F0 0F     106          BEQ  BDEL40     OTHERWISE END SEARCH
9E54:A0 00     107 BDEL35   LDY  #0          POINT TO NEXT LINE
9E56:B1 42     108          LDA  (START),Y
9E58:AA        109          TAX
9E59:C8        110          INY
9E5A:B1 42     111          LDA  (START),Y
9E5C:85 43     112          STA  START+1
9E5E:86 42     113          STX  START
```

```
9E60:4C 3C 9E  114          JMP  BDEL25    LOOP BACK & KEEP SEARCHING
9E63:          115 *
9E63:A0 00     116 BDEL40   LDY  #0        POINT TO NEXT LINE AFTER END LINE#
9E65:B1 42     117          LDA  (START),Y
9E67:AA        118          TAX
9E68:C8        119          INY
9E69:B1 42     120          LDA  (START),Y
9E6B:85 43     121          STA  START+1
9E6D:86 42     122          STX  START
9E6F:          123 *
9E6F:38        124          SEC            CALCULATE DIFFERENCE IN ADDRESSES
9E70:A5 40     125          LDA  DEST
9E72:48        126          PHA
9E73:E5 42     127          SBC  START
9E75:85 44     128          STA  ADJUST
9E77:A5 41     129          LDA  DEST+1
9E79:48        130          PHA
9E7A:E5 43     131          SBC  START+1
9E7C:85 45     132          STA  ADJUST+1
9E7E:          133 *
9E7E:20 B4 9E  134          JSR  MOVE      OVERWRITE LINES TO BE DELETED
9E81:          135 *
9E81:68        136          PLA            INITIALISE EOPP FOR CALCULATING
9E82:85 9D     137          STA  EOPP+1    NEW LINK VALUES
9E84:68        138          PLA
9E85:85 9C     139          STA  EOPP
9E87:          140 *
9E87:A0 00     141 BDEL45   LDY  #0        FIX POINTERS TO GIVE NEW ADDRESSES
9E89:B1 9C     142          LDA  (EOPP),Y  END OF PROGRAM?
9E8B:D0 05     143          BNE  BDEL50    NO? THEN SKIP
9E8D:C8        144          INY
9E8E:B1 9C     145          LDA  (EOPP),Y
9E90:F0 18     146          BEQ  BDEL55    OTHERWISE SET NEW EOPP
9E92:A0 00     147 BDEL50   LDY  #0        ADJUST LINK TO NEXT RECORD
9E94:18        148          CLC
9E95:B1 9C     149          LDA  (EOPP),Y
9E97:65 44     150          ADC  ADJUST
9E99:91 9C     151          STA  (EOPP),Y
9E9B:AA        152          TAX
9E9C:C8        153          INY
9E9D:B1 9C     154          LDA  (EOPP),Y
9E9F:65 45     155          ADC  ADJUST+1
9EA1:91 9C     156          STA  (EOPP),Y
9EA3:85 9D     157          STA  EOPP+1    & POINT TO NEXT RECORD
9EA5:86 9C     158          STX  EOPP
9EA7:4C 87 9E  159          JMP  BDEL45
9EAA:          160 *
9EAA:20 AD 9E  161 BDEL55   JSR  BDEL60    ADD 2 TO EOPP
9EAD:          162 *
9EAD:E6 9C     163 BDEL60   INC  EOPP
9EAF:D0 02     164          BNE  BDEL99
9EB1:E6 9D     165          INC  EOPP+1
9EB3:          166 *
9EB3:60        167 BDEL99   RTS

9EB4:          169 ********************************
9EB4:          170 *
9EB4:          171 * MOVE - MOVE MEMORY DOWN
```

```
9EB4:          172 *

9EB4:A0 00     174 MOVE    LDY  #0
9EB6:          175 *
9EB6:B1 42     176 MOVE01  LDA  (START),Y MOVE BYTE
9EB8:91 40     177         STA  (DEST),Y
9EBA:          178 *
9EBA:E6 40     179         INC  DEST      NEXT LOCATION TO MOVE TO
9EBC:D0 02     180         BNE  MOVE02
9EBE:E6 41     181         INC  DEST+1
9EC0:A5 42     182 MOVE02  LDA  START     FINISHED?
9EC2:C5 9C     183         CMP  EOPP
9EC4:D0 06     184         BNE  MOVE04
9EC6:A5 43     185         LDA  START+1
9EC8:C5 9D     186         CMP  EOPP+1
9ECA:F0 09     187         BEQ  MOVE05
9ECC:          188 *
9ECC:E6 42     189 MOVE04  INC  START     NEXT LOCATION TO MOVE FROM
9ECE:D0 E6     190         BNE  MOVE01
9ED0:E6 43     191         INC  START+1
9ED2:4C B6 9E  192         JMP  MOVE01
9ED5:          193 *
9ED5:          194 *
9ED5:60        195 MOVE05  RTS



SYMBOL TABLE    SORTED BY SYMBOL

   44 ADJUST       ?9E00 BDEL00        9E09 BDEL05        9E17 BDEL10
 9E21 BDEL15        9E30 BDEL20        9E3C BDEL25        9E48 BDEL28
 9E4A BDEL30        9E54 BDEL35        9E63 BDEL40        9E87 BDEL45
 9E92 BDEL50        9EAA BDEL55        9EAD BDEL60        9EB3 BDEL99
   40 DEST          0402 ENDL            9C EOPP          9EB4 MOVE
 9EB6 MOVE01        9EC0 MOVE02        9ECC MOVE04        9ED5 MOVE05
   9A SOPP          0400 STARTL          42 START

SYMBOL TABLE    SORTED BY ADDRESS

   40 DEST            42 START           44 ADJUST          9A SOPP
   9C EOPP          0400 STARTL        0402 ENDL          ?9E00 BDEL00
 9E09 BDEL05        9E17 BDEL10        9E21 BDEL15        9E30 BDEL20
 9E3C BDEL25        9E48 BDEL28        9E4A BDEL30        9E54 BDEL35
 9E63 BDEL40        9E87 BDEL45        9E92 BDEL50        9EAA BDEL55
 9EAD BDEL60        9EB3 BDEL99        9EB4 MOVE          9EB6 MOVE01
 9EC0 MOVE02        9ECC MOVE04        9ED5 MOVE05
```

Graphics Aid



This utility program allows for interactive creation of 'personalized' characters. The user is able to redefine any character from the alternative character set. Since the usage of 1 character to generate graphics characters is limited by its small dimensions, the utility incorporates an expansion system whereby the user can redefine a 2x2 character grid (=4 char). This is simply an attempt to overcome the limitations imposed by a 1 character redefinition if one wishes to redefine more than one character to create an object.

The utility prompts the user for all information. When designing the 'graphic creation' the user needs only position the cursor anywhere inside the grid and turn on/off the corresponding (enlarged) pixel. For the user's convenience an 'actual size' pattern is plotted to the right of the enlarged grid.

All characters are updated 'real time'. This is why the alternative character set is used. If the standard set were used then normal text would be affected. (Warning: reset will return the character set to its normal state).

An added feature of the utility is that it is able to output a 'DATAMAP' of the characters being redefined. The user needs then only copy the statements and incorporate them into his own program. Filling/blanking is also available.

We used this program to create most of our moving craft in the games in this book. Even if you wanted to use the alternate characters you would still have plenty left to redefine, as many alternate characters are duplicated several times within the set. However there is no problem redefining the standard set and if you really want to, go ahead.

To use the standard character set change the address (4194) in line 1320 to 46080. Remove line 2000 and take the CHR$(9) out of lines 1100 & 1110.

DRIVING THE UTILITY:

Expansion factor?
1 = 1x1 CHAR GRID
2 = 2x2 CHAR GRID

Utility will need to know character(s) to redefine and will prompt for it/them.

This accomplished the utility plots the grids (Large/actual) and then waits for new input. Press:

right arrow	to move right
left arrow	to move left
up arrow	to move up
down arrow	to move down
space bar	to turn ON/OFF a pixel (if pixel is on before pressing then pixel will go off and vice versa)
return on ESC	will end graphic creation

Utility now prompts user as to whether he wants a datamap followed by a prompt to end/start again fresh/continue on.

```
  0 REM * GRAPHICS UTILITY *
  1 DIMSC(12 , 16) , CM(12 , 16) , CH$(4)
  2 DIM CSA(4)
  4 PAPER 6 : INK 0
  5 CLS
  6 CF=1
 10  : PRINT "      *** GRAPHICS UTILITY ***"
 11  : PRINT :  PRINT
 20 INPUT " Number of characters(1 or 4) -> ";SZ
 30 IF SZ=1 OR SZ=4 GOTO60
 40 PRINT "*** ERROR - PLEASE ENTER 1 OR 4"
 50  :  WAIT 100  :  GOTO 2
 60 IF SZ<>4 THEN 70
 61 GOSUB 1000
 62 GOSUB 2000
 63 GOTO 80
 70 PRINT :  PRINT " ENTER CHARACTER TO REDEFINE ";
 71 GET CH$(1)  :  PRINT CH$(1)
 72 PRINT : PRINT " Redefining ";"'";CH$(1);"'"
 73 PRINT : PRINT " Is this correct ('E' TO ERASE
    FIRST) ?";
 74 GET A$ : IF A$="E"THEN EE=1 : GOSUB2000 : GOTO80
 75 IF A$="Y" THEN PRINT A$ :  GOSUB 2000 : GOTO 80
 76 IF A$<>"N" THEN 74
 77 GOTO 70
 79 GOSUB 2000
 80 OF=20-CO : FOR I=1TOOF : PRINTDC$; : NEXT
 81 PRINT "PRINT DATA MAP (Y/N)";
 90  :  GET A$  :  PRINT A$
100 IF A$="Y" THEN GOSUB 6000
101 FOR I=1TORW : PRINT RC$; : NEXT
110  :  PRINT "MORE EDITING ?";
120  :  GET A$ : PRINT A$
130  :  IF A$="Y" THEN 60
140  :  IF A$<>"N" THEN 120
150  :  END
200 GOSUB1000 : GOTO80
520 REM *  Design using 4 chars
560 REM *** CURSOR MANIPULATION ROUTINES
1000 PRINT :  PRINT "PLEASE SPECIFY THE 4 CHARACTERS
     YOU WISH TO USE"
1010 PRINT : PRINT "UPPER  LEFT  : ";
1020 GET CH$(1) : PRINT CH$(1)
1030 PRINT "UPPER RIGHT : ";
1040 GET CH$(2) : PRINT CH$(2)
1050 PRINT "LOWER  LEFT : ";
1060 GET CH$(3) : PRINT CH$(3)
1070 PRINT "LOWER  RIGHT : ";
1080 GET CH$(4) : PRINT CH$(4)
1090 PRINT : PRINT : PRINT
1100 PLOT2 , 13 , CHR$(9)+CH$(1) + CH$(2)
1110 PLOT2 , 14 , CHR$(9)+CH$(3) + CH$(4)
1111 PRINT
```

```
1112 EF=0
1120 FOR I= 1 TO 3
1121 M=I+1
1130 FOR J= M TO 4
1140 IF CH$(I) = CH$(J) THEN EF=I
1150 NEXT J
1160 NEXT I
1170 IF EF= 0 GOTO 1200
1180 PRINT "*** ERROR -> CHARS MUST BE UNIQUE"
1190 PRINT "**** DUPLICATE ";CH$(EF);"'s" : GOTO 1000
1192 REM *** Verify the characters
1200 PRINT "Is this correct ('E' TO ERASE FIRST ) ? ";
1210 GET A$
1220 IF A$="N" THEN 1000
1225 IF A$="E"THEN EE=1 : GOTO 1231
1229 EE=0
1230 IF A$<> "Y" THEN 1210
1231 PRINT A$ : PRINT : PRINT " Please wait"
1234 REM  SET UP CHAR MAP OF 2*2 GRID
1235 REM  QUADRANTS  E.G 1 2
1236 REM                 3 4
1237 REM  1 QUADRANT OF ARRAY WILL
1238 REM  CONTAIN ONE OF THE ABOVE
1239 REM  VALUES. 1'S IN TOP LHS ETC.
1240 FOR J= 1 TO 16
1250 FOR I= 1 TO 12
1260 N= INT(J/9)*2 +INT(I/7)+1
1270 CM(I , J)=N
1280 NEXT I , J
1294 REM   Calculate addresses of
1295 REM   redefined chars. Store
1296 REM   in CSA array.
1300 FOR I= 1 TO 4
1310 N=ASC(CH$(I))
1320 CSA(I)= 47104 + (N*8)
1330 NEXT I
1331 IFA$<>"E"THEN RETURN
1333 FOR I=1 TO 4 : FORK=CSA(I)TOCSA(I)+7 : POKE K ,
     0 : NEXT K , I
1340 RETURN
1990 ***   2 X 2   GRID    ***
2000 CLS
2001 LI$="+ "
2002 RO=4 : CL=7  :  RW=1 : CO=1
2003 IFSZ=1THENLC=6ELSELC=12
2004 IFSZ=1THENLR=8ELSELR=16
2005 MR=LR/2+4
2006 GOSUB7000
2010 PLOT LC+15 , MR , CH$(1)
2011 PLOT LC+16 , MR , CH$(2)
2012 PLOT LC+15 , MR+1 , CH$(3)
2013 PLOT LC+16 , MR+1 , CH$(4)
2020 PLOT CL , RO , LI$
2030 PLOT CL+LC+1 , RO , LI$
2040 PLOT CL , RO+LR-1 , LI$
2050 PLOT CL+LC+1 , RO+LR-1 , LI$
```

```
2053 REM      Set cursor characters
2060 UC$= CHR$(11)       : REM UP
2070 DC$= CHR$(10)       : REM DOWN
2080 LC$= CHR$(8)        : REM LEFT
2090 RC$= CHR$(9)        : REM RIGHT
2095 IF EE=1 THEN 2105
2100 GOSUB 5000
2105 FOR I= 1 TO 4 : PRINT DC$; : NEXT
2110 FOR I= 1 TO 9 : PRINT RC$; : NEXT
2120 IF CF<>0 GOTO 2300
2130 FOR I= 1 TO LC
2140 FOR J= 1 TO LR
2142 SC(I , J) = 0
2146 NEXT J
2148 NEXT I
2150 REM
2200 REM
2210 REM Get a character and DEF  then
2230 REM determine appropriate action
2300 KP$= KEY$ :  IF KP$ ="" THEN 2300
2400 K= ASC(KP$)
2410 IF K<12 AND K>7 THEN 2610
2420 IF K= 32 THEN 2460
2430 IF K= 27 OR K= 13 THEN RETURN
2440 PRINT CHR$(7); : GOTO 2300
2452 REM    Space bar depressed
2460 IF SC(CO , RW) = 1  THEN 2500
2470 SC(CO , RW) = 1
2475 SW= 1 : GOSUB 4000 : REM PIX ON
2480 PLOT CO+7 , RW+3 , CHR$(95)
2490 GOTO 2300
2500 SC(CO , RW) = 0
2505 SW= 0 : GOSUB 4000 : REM PIX OFF
2510 PLOT CO+7 , RW+3 , " "
2520 GOTO 2300
2570 REM  Cursor movement routines
2580 REM  follow- up , down , left , right
2610 K= K - 7  : REM MAKE RANGE 1 -> 4
2620 ON K GOTO 2630 , 3000 , 3200 , 3300
2630 REM ***  L e f t  C u r s o r
2640 IF CO>1 THENCO= CO-1  : PRINT LC$;
2650 GOTO2300
3000 REM *** R i g h t  C u r s o r
3002 IF CO<>LC THEN CO=CO+1 : PRINT RC$;
3010 GOTO2300
3200 REM ***  D o w n  C u r s o r
3210 IF RW < LR THEN RW= RW+1 : PRINT DC$;
3220 GOTO 2300
3300 REM ***  U p  C u r s o r
3310 IFRW > 1 THEN RW=RW-1 : PRINT UC$;
3320 GOTO2300
4000 CH= CM(CO , RW)    : REM GET CHAR
4010 SA= CSA(CH)      : REM RAM ADDRESS
4020 IF RW<9 THEN BY=RW-1 :  GOTO 4040
4030 BY=RW-9
4040 IF CO<7 THEN BT= ABS(CO-6) : GOTO 4100
```

```
4050 BT= ABS(CO-12)
4090 REM    BY -> HOLDS BYTE OFFSET
4091 REM    BT -> HOLDS BIT TO TOGGLE
4100 SA= SA + BY      : REM START+OFFSET
4101 IF SW= 1 GOTO 4200 : REM OFF/ON?
4110 REM
4111 REM *** Turn off the pixel
4112 REM
4120 K= 255 - (2 ^ BT)  : REM SET MASK
4130 POKE SA , PEEK(SA) AND K : REM OFF!!
4140 RETURN
4200 K= (2 ^ BT)        : REM SET MASK
4210 POKE SA , PEEK(SA) OR K : REM  ON!!
4220 RETURN
5000 FOR K=1 TO SZ
5010 SA=CSA(K)
5020 FOR I= 0 TO 7
5030 NUM= PEEK (SA + I) : REM CHR BYTE
5040 FOR J= 5 TO 0 STEP -1
5050 KN= NUM AND (2 ^ J)
5060 IF KN <> 0 THEN KN= 1  : REM ON!
5062 REM ****************************
5064 REM CALCULATE RO+ & COLUMN
5067 AB=ABS(J-5)
5070 CO= AB+1+6*(ABS(K=2)ORABS(K=4))
5080 RW= I+1+8*(ABS(K=3)ORABS(K=4))
5090 SC(CO , RW)= KN   : REM MAP BIT
5100 IF KN=0 GOTO 5120
5110 PLOT CO+7 , RW+3 , CHR$(95)
5120 NEXT J
5130 NEXT I
5140 NEXT K
5150 RW=1 : CO=1
5160 RETURN
6000 REM PRINT CHARACTER DATA
6010 PRINT
6020 FOR K=1 TO SZ
6025 PRINT"DATA ";
6030 FOR I=CSA(K)TOCSA(K)+7
6040 PRINT PEEK(I);",";
6050 NEXT I : PRINT : PRINT : NEXT K  : PRINT : PRINT
6090 RETURN
7000 FORI=ROTORO+LR-1 : PLOTO , I , CHR$(9) : NEXTI
7010 POKE47448 , 8 : POKE47449 , 8 : POKE47450 , 8 :
     POKE47451 , 63
7020 POKE47452 , 8 : POKE47453 , 8 : POKE47454 , 8 :
     POKE47455 , 0
7030 FORI=#BB78TO#BB7F : POKEI , 63 : NEXTI
7040 RETURN
```

WARNING

Your Oric-1 is about to become Meteoric.

Now you can achieve Meteoric software performance from your micro with this collection of 30 dynamic programs, designed specifically for the Oric-1 home computer.

You can test your skill with arcade action games such as Luna Landa, Eliminator and North Sea Copter, or attempt to solve the mind stretching mathematical and educational exercises.

These programs fully extend the excellent colour resolution of the Oric-1 to its maximum!

The book includes many practical utilities that will enable you to really improve your own programs. These useful routines include Renumber, Machine Code Monitor, Block Line Deletion and Graphics Aid.

Even if you are a beginner, you can enter all the programs yourself due to Meteoric programming's easy to understand format.

Your overall knowledge of computer operation will be expanded too. Each program features an in-depth explanation of how and why it runs and illustrations of screen display back up the text to ensure you understand exactly what will be achieved.

Experienced programmers who want to develop their own Meteoric programs will really appreciate all the expert hints and tips that have been included in the book.

Whether you're a first time user or an old hand, you can't afford to miss the only book that will make your Oric-1 Meteoric.

ISBN 0 86161 1241

£5·95

METEORIC PROGRAMMING